Enrique Cabello
Jorge Cardoso
Leszek A. Maciaszek
Marten van Sinderen (Eds.)

Communications in Computer and Information Science 868

Software Technologies

12th International Joint Conference, ICSOFT 2017
Madrid, Spain, July 24–26, 2017
Revised Selected Papers

Springer

Communications in Computer and Information Science 868

Enrique Cabello · Jorge Cardoso
Leszek A. Maciaszek · Marten van Sinderen (Eds.)

Software Technologies

12th International Joint Conference, ICSOFT 2017
Madrid, Spain, July 24–26, 2017
Revised Selected Papers

Springer

Editors
Enrique Cabello
King Juan Carlos University
Madrid
Spain

Jorge Cardoso
University of Coimbra
Coimbra
Portugal

Leszek A. Maciaszek
Wroclaw University of Economics
Wroclaw
Poland

Marten van Sinderen
Computer Science
University of Twente
Enschede
The Netherlands

ISSN 1865-0929 ISSN 1865-0937 (electronic)
Communications in Computer and Information Science
ISBN 978-3-319-93640-6 ISBN 978-3-319-93641-3 (eBook)
https://doi.org/10.1007/978-3-319-93641-3

Library of Congress Control Number: 2018947013

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG
part of Springer Nature
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The present book includes extended and revised versions of a set of selected papers from the 12th International Conference on Software Technologies (ICSOFT 2017), held in Madrid, Spain, during July 24–26.

ICSOFT 2017 received 85 paper submissions from 33 countries, of which 15% are included in this book. The papers were selected by the event chairs and their selection is based on a number of criteria that include the classifications and comments provided by the Program Committee members, the session chairs' assessment, and also the program chairs' perception of the overall quality of papers included in the technical program. The authors of selected papers were then invited to submit a revised and extended version of their papers having at least 30% innovative material.

The purpose of the ICSOFT conferences, including its 12th edition in 2017, is to bring together researchers and practitioners interested in developing and using software technologies for the benefit of businesses and society at large. The conference solicits papers and other contributions in themes ranging from software engineering and development via showcasing cutting-edge software systems and applications to addressing foundational innovative technologies for systems and applications of the future.

The papers selected to be included in this book conform to the ICSOFT purpose and contribute to the understanding of current research and practice on software technologies. The main topics covered in the papers include: software quality and metrics (Chaps. 1, 2, 6 and 9), software testing and maintenance (Chap. 2), development methods and models (Chaps. 3, 4, 5 and 9), systems security (Chap. 6), dynamic software updates (Chap. 7), systems integration (Chap. 8), business process modelling (Chap. 9), intelligent problem solving (Chap. 10), multi-agent systems (Chap. 12), and solutions involving big data, the Internet of Things and business intelligence (Chaps. 11 and 13).

We would like to thank all the authors for their contributions and the reviewers for ensuring the quality of this publication.

July 2017

Enrique Cabello
Jorge Cardoso
Leszek Maciaszek
Marten van Sinderen

Organization

Conference Chair

Enrique Cabello	Universidad Rey Juan Carlos, Spain

Program Co-chairs

Jorge Cardoso	University of Coimbra, Portugal and Huawei German Research Center, Munich, Germany
Leszek Maciaszek	Wroclaw University of Economics, Poland and Macquarie University, Sydney, Australia
Marten van Sinderen	University of Twente, The Netherlands

Program Committee

Markus Aleksy	ABB Corporate Research Center, Germany
Waleed Alsabhan	KACST, UK
Bernhard Bauer	University of Augsburg, Germany
Maurice H. ter Beek	ISTI-CNR, Pisa, Italy
Wolfgang Bein	University of Nevada, Las Vegas, USA
Fevzi Belli	Izmir Institute of Technology, Turkey
Gábor Bergmann	Budapest University of Technology and Economics, Hungary
Mario Luca Bernardi	Giustino Fortunato University, Italy
Jorge Bernardino	Polytechnic Institute of Coimbra, ISEC, Portugal
Mario Berón	Universidad Nacional de San Luis, Argentina
Marcello M. Bersani	Politecnico di Milano, Italy
Thomas Buchmann	University of Bayreuth, Germany
Miroslav Bureš	Czech Technical University, Czech Republic
Nelio Cacho	Federal University of Rio Grande do Norte, Brazil
Antoni Lluís Mesquida Calafat	Universitat de les Illes Balears, Spain
Jose Antonio Calvo-Manzano	Universidad Politécnica de Madrid, Spain
Ana R. Cavalli	Institute Telecom SudParis, France
Marta Cimitile	Unitelma Sapienza, Italy
Felix J. Garcia Clemente	University of Murcia, Spain
Kendra Cooper	Independent Scholar, Canada
Agostino Cortesi	Università Ca' Foscari di Venezia, Italy
António Miguel Rosado da Cruz	Instituto Politécnico de Viana do Castelo, Portugal
Lidia Cuesta	Universitat Politècnica de Catalunya, Spain

Luis Fernandez Sanz	University of Alcala, Spain
Elad Michael Schiller	Chalmers University of Technology, Sweden
Istvan Siket	Hungarian Academy of Science, Research Group on Artificial Intelligence, Hungary
Michal Smialek	Warsaw University of Technology, Poland
Cosmin Stoica Spahiu	University of Craiova, Romania
Miroslaw Staron	University of Gothenburg, Sweden
Anca-Juliana Stoica	Uppsala University, Sweden
Ketil Stølen	SINTEF, Norway
Hiroki Suguri	Miyagi University, Japan
Bedir Tekinerdogan	Wageningen University, The Netherlands
Chouki Tibermacine	LIRMM, CNRS and Montpellier University, France
Claudine Toffolon	Université du Maine, France
Michael Vassilakopoulos	University of Thessaly, Greece
Dessislava Vassileva	Sofia University St. Kliment Ohridski, Bulgaria
László Vidács	University of Szeged, Hungary
Sergiy Vilkomir	East Carolina University, USA
Gianluigi Viscusi	EPFL Lausanne, Switzerland
Christiane Gresse von Wangenheim	Federal University of Santa Catarina, Brazil
Dietmar Winkler	Vienna University of Technology, Austria
Dianxiang Xu	Boise State University, USA
Jinhui Yao	Xerox Research, USA
Murat Yilmaz	Çankaya University, Turkey
Jingyu Zhang	Macquarie University, Australia

Additional Reviewers

Doina Bein	California State University, Fullerton, USA
Dominik Bork	University of Vienna, Austria
Angela Chan	University of Nevada, Reno, USA
Estrela Ferreira Cruz	Instituto Politécnico de Viana do Castelo, Portugal
Alessandro Fantechi	University of Florence, Italy
Dusan Gajic	University of Novi Sad, Serbia
Jalal Kiswani	University of Nevada, Reno, USA
Asia van de Mortel-Fronczak	Eindhoven University of Technology, The Netherlands
Benedikt Pittl	University of Vienna, Austria
Fredrik Seehusen	Sintef, Norway
Rocky Slavin	University of Texas at San Antonio, USA
Gábor Szárnyas	Budapest University of Technology and Economics, Hungary
Michael Walch	University of Vienna, Austria

Invited Speakers

Jan Bosch	Chalmers University of Technology, Sweden
Siobhán Clarke	Trinity College Dublin, Ireland
Stefano Ceri	Politecnico di Milano, Italy
Andreas Holzinger	Medical University Graz, Austria

Contents

Software Engineering

Assessing the User-Perceived Quality of Source Code Components Using Static Analysis Metrics

Valasia Dimaridou, Alexandros-Charalampos Kyprianidis, Michail Papamichail, Themistoklis Diamantopoulos(✉), and Andreas Symeonidis

Electrical and Computer Engineering Department, Aristotle University of Thessaloniki, Thessaloniki, Greece
{valadima,alexkypr}@ece.auth.gr, {mpapamic,thdiaman}@issel.ee.auth.gr, asymeon@eng.auth.gr

Abstract. Nowadays, developers tend to adopt a component-based software engineering approach, reusing own implementations and/or resorting to third-party source code. This practice is in principle cost-effective, however it may also lead to low quality software products, if the components to be reused exhibit low quality. Thus, several approaches have been developed to measure the quality of software components. Most of them, however, rely on the aid of experts for defining target quality scores and deriving metric thresholds, leading to results that are context-dependent and subjective. In this work, we build a mechanism that employs static analysis metrics extracted from GitHub projects and defines a target quality score based on repositories' stars and forks, which indicate their adoption/acceptance by developers. Upon removing outliers with a one-class classifier, we employ Principal Feature Analysis and examine the semantics among metrics to provide an analysis on five axes for source code components (classes or packages): complexity, coupling, size, degree of inheritance, and quality of documentation. Neural networks are thus applied to estimate the final quality score given metrics from these axes. Preliminary evaluation indicates that our approach effectively estimates software quality at both class and package levels.

Keywords: Code quality · Static analysis metrics
User-perceived quality · Principal Feature Analysis

1 Introduction

The continuously increasing need for software applications in practically every domain, and the introduction of online open-source repositories have led to the establishment of an agile, component-based software engineering paradigm. The need for reusing existing (own or third-party) source code, either in the form of software libraries or simply by applying copy-paste-integrate practices has

E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 3–27, 2018.
https://doi.org/10.1007/978-3-319-93641-3_1

become more eminent than ever, since it can greatly reduce the time and cost of software development [19]. In this context, developers often need to spend considerable time and effort to integrate components and ensure high performance. And still, this may lead to failures, since the reused code may not satisfy basic functional or non-functional requirements. Thus, the quality assessment of reusable components poses a major challenge for the research community.

An important aspect of this challenge is the fact that quality is context-dependent and may mean different things to different people [17]. Hence, a standardized approach for measuring quality has been proposed in the latest ISO/IEC 25010:2011 [10], which defines a model with eight quality characteristics: Functional Suitability, Usability, Maintainability, Portability, Reliability, Performance and Efficiency, Security and Compatibility, out of which the first four are usually assessed using static analysis and evaluated intuitively by developers. To accommodate reuse, developers usually structure their source code (or assess third-party code) so that it is modular, exhibits loose coupling and high cohesion, and provides information hiding and separation of concerns [16].

Current research efforts assess the quality of software components using static analysis metrics [4,12,22,23], such as the known CK metrics [3]. Although these efforts can be effective for the assessment of a quality characteristic (e.g. [re]usability, maintainability or security), they do not actually provide an interpretable analysis to the developer, and thus do not inform him/her about the source code properties that need improvement. Moreover, the approaches that are based on metric thresholds, whether defined manually [4,12,23] or derived automatically using a model [24], are usually constrained by the lack of objective ground truth values for software quality. As a result, these approaches typically resort to expert help, which may be subjective, case-specific or even unavailable [2]. An interesting alternative is proposed by Papamichail et al. [15] that employ user-perceived quality as a measure of the quality of a software component.

In this work, we employ the concepts defined in [15] and build upon the work originated from [5], which performs analysis only at class level, in order to build a mechanism that associates the extent to which a software component (class or package) is adopted/preferred by developers. We define a ground truth score for the user-perceived quality of components based on popularity-related information extracted from their GitHub repos, in the form of stars and forks. Then, at each level, we employ a one-class classifier and build a model based on static analysis metrics extracted from a set of popular GitHub projects. By using Principal Feature Analysis and examining the semantics among metrics, we provide the developer with not only a quality score, but also a comprehensive analysis on five axes for the source code of a component, including scores on its complexity, coupling, size, degree of inheritance, and the quality of its documentation. Finally, for each level, we construct five Neural Networks models, one for each of these code properties, and aggregate their output to provide an overall quality scoring mechanism at class and package level, respectively.

The rest of this paper is organized as follows. Section 2 provides background information on static analysis metrics and reviews current approaches on quality estimation. Section 3 describes our benchmark dataset and designs a scoring mechanism for the quality of source code components. The constructed models are shown in Sect. 4, while Sect. 5 evaluates the performance of our system. Finally, Sect. 6 concludes this paper and provides insight for further research.

2 Related Work

According to [14], research on software quality is as old as software development. As software penetrates everyday life, assessing quality has become a major challenge. This is reflected in the various approaches proposed by current literature that aspire to assess quality in a quantified manner. Most of these approaches make use of static analysis metrics in order to train quality estimation models [12,18]. Estimating quality through static analysis metrics is a non-trivial task, as it often requires determining quality thresholds [4], which is usually performed by experts who manually examine the source code [8]. However, the manual examination of source code, especially for large complex projects that change on a regular basis, is not always feasible due to constraints in time and resources. Moreover, expert help may be subjective and highly context-specific.

Other approaches may require multiple parameters for constructing quality evaluation models [2], which are again highly dependent on the scope of the source code and are easily affected by subjective judgment. Thus, a common practice involves deriving metric thresholds by applying machine learning techniques on a benchmark repository. Ferreira et al. [6] propose a methodology for estimating thresholds by fitting the values of metrics into probability distributions, while [1] follow a weight-based approach to derive thresholds by applying statistical analysis on the metrics values. Other approaches involve deriving thresholds using bootstrapping [7] and ROC curve analysis [20]. Still, these approaches are subject to the projects selected for the benchmark repository.

An interesting approach that refrains from the need to use certain metrics thresholds and proposes a fully automated quality evaluation methodology is that of Papamichail et al. [15]. The authors design a system that reflects the extent to which a software component is of high quality as perceived by developers. The proposed system makes use of crowdsourcing information (the popularity of software projects) and a large set of static analysis metrics, in order to provide a single quality score, which is computed using two models: a one-class-classifier used to identify high quality code and a neural network that translates the values of the static analysis metrics into quantified quality estimations.

Although the aforementioned approaches can be effective for certain cases, their applicability in real-world scenarios is limited. The use of predefined thresholds [4,8] results in the creation of models unable to cover the versatility of today's software, and thus applies only to restricted scenarios. On the other hand, systems that overcome threshold issues by proposing automated quality evaluation methodologies [15] often involve preprocessing steps (such as feature extraction) or regression models that lead to a quality score which is not interpretable. As a result, the developer is provided with no specific information on the targeted changes to apply in order to improve source code quality.

Extending previous work [5], we have built a generic source code quality estimation mechanism able to provide a quality score at both class and package levels, which reflects the extent to which a component could/should be adopted by developers. Our system refrains from expert-based knowledge and employs a large set of static analysis metrics and crowdsourcing information from GitHub stars and forks in order to train five quality estimation models for each level, each one targeting a different property of source code. The individual scores are then combined to produce a final quality score that is fully interpretable and provides necessary information towards the axes that require improvement. By further analyzing the correlation and the semantics of the metrics for each axis, we are able to identify similar behaviors and thus select the ones that accumulate the most valuable information, while at the same time describing the characteristics of the source code component under examination.

3 Defining Quality

In this section, we quantify quality as perceived by developers using information from GitHub stars and forks as ground truth. In addition, our analysis describes how the different categories of source code metrics are related to major quality characteristics as defined in ISO/IEC 25010:2011 [10].

3.1 Benchmark Dataset

Our dataset consists of a large set of static analysis metrics calculated for 102 repositories, selected from the 100 most starred and the 100 most forked GitHub Java projects. The projects were sorted in descending order of stars and subsequently forks, and were selected to cover more than 100,000 classes and 7,300 projects. Certain statistics of the benchmark dataset are shown in Table 1.

Table 1. Dataset statistics [5].

Statistics	Dataset
Total number of projects	102
Total number of packages	7,372
Total number of classes	100,233
Total number of methods	584,856
Total lines of code	7,985,385

We compute a large set of static analysis metrics that cover the source code properties of complexity, coupling, documentation, inheritance, and size. Current literature [9,11] indicates that these properties are directly related to the characteristics of Functional Suitability, Usability, Maintainability, and Portability, as defined by ISO/IEC 25010:2011 [10]. The metrics that were computed

Table 2. Overview of static metrics and their applicability on different levels.

Static analysis metrics			Compute levels	
Type	Name	Description	Class	Package
Complexity	NL	Nesting Level	×	
	NLE	Nesting Level Else-If	×	
	WMC	Weighted Methods per Class	×	
Coupling	CBO	Coupling Between Object classes	×	
	CBOI	CBO Inverse	×	
	NII	Number of Incoming Invocations	×	
	NOI	Number of Outgoing Invocations	×	
	RFC	Response set For Class	×	
Cohesion	LCOM5	Lack of Cohesion in Methods 5	×	
Documentation	AD	API Documentation	×	
	CD	Comment Density	×	×
	CLOC	Comment Lines of Code	×	×
	DLOC	Documentation Lines of Code	×	
	PDA	Public Documented API	×	×
	PUA	Public Undocumented API	×	×
	TAD	Total API Documentation		×
	TCD	Total Comment Density	×	×
	TCLOC	Total Comment Lines of Code	×	×
	TPDA	Total Public Documented API		×
	TPUA	Total Public Undocumented API		×
Inheritance	DIT	Depth of Inheritance Tree	×	
	NOA	Number of Ancestors	×	
	NOC	Number of Children	×	
	NOD	Number of Descendants	×	
	NOP	Number of Parents	×	
Size	{L}LOC	{Logical} Lines of Code	×	×
	N{A, G, M, S}	Number of {Attributes, Getters, Methods, Setters}	×	×
	N{CL, EN, IN, P}	Number of {Classes, Enums, Interfaces, Packages}		×
	NL{A, G, M, S}	Number of Local {Attributes, Getters, Methods, Setters}	×	
	NLP{A, M}	Number of Local Public {Attributes, Methods}	×	
	NP{A, M}	Number of Public {Attributes, Methods}	×	×
	NOS	Number of Statements	×	
	T{L}LOC	Total {Logical} Lines of Code	×	×
	TNP{CL, EN, IN}	Total Number of Public {Classes, Enums, Interfaces}		×
	TN{CL, DI, EN, FI}	Total Number of {Classes, Directories, Enums, Files}		×

using SourceMeter [21] are shown in Table 2. In our previous work [5], the metrics were computed at class level, except for McCC that was computed at method level and then averaged to obtain a value for the class. For this extended work the metrics were computed at a package level, except for the metrics that are available only at class level. These metrics were initially calculated at class level and the median of each one was enumerated to obtain values for the packages.

3.2 Quality Score Formulation

As already mentioned, we use GitHub stars and forks as ground truth information towards quantifying quality as perceived by developers. According to our initial hypothesis, the number of stars can be used as a measure of the popularity for a software project, while the number of forks as a measure of its reusability. We make use of this information in order to define our target variable and consequently build a quality scoring mechanism. Towards this direction, we aim to define a quality score for every class and every package included in the dataset.

Given, however, that the number of stars and forks refer to repository level, they are not directly suited for defining a score that reflects the quality of each class or package, individually. Obviously, equally splitting the quality score computed at repository level among all classes or packages is not optimal, as every component has a different significance in terms of functionality and thus must be rated as an independent entity. Consequently, in an effort to build a scoring mechanism that is as objective as possible, we propose a methodology that involves the values of static analysis metrics for modeling the significance of each source code component (class or package) included in a given repository.

The quality score for every software component (class or package) of the dataset is defined using the following equations:

$$S_{stars}(i,j) = (1 + NPM(j)) \cdot \frac{Stars(i)}{N_{components}(i)} \tag{1}$$

$$S_{forks}(i,j) = (1 + AD(j) + NM(j)) \cdot \frac{Forks(i)}{N_{components}(i)} \tag{2}$$

$$Q_{score}(i,j) = \log(S_{stars}(i,j) + S_{forks}(i,j)) \tag{3}$$

where $S_{stars}(i,j)$ and $S_{forks}(i,j)$ represent the quality scores for the j-th source code component (class or package) contained in the i-th repository, based on the number of GitHub stars and forks, respectively. $N_{components}(i)$ corresponds to the number of source code components (classes or packages) contained in the i-th repository, while $Stars(i)$ and $Forks(i)$ refer to the number of its GitHub stars and forks, respectively. Finally, $Q_{score}(i,j)$ is the overall quality score computed for the j-th source code component (class or package) contained in the i-th repository.

Our target set also involves the values of three metrics as a measure of the significance for every individual class or package contained in a given repository. Different significance implies different contribution to the number of GitHub stars and forks of the repository and thus different quality scores. $NPM(j)$ is used to measure the degree to which the j-th class (or package) contributes to the number of stars of the repository, as it refers to the number of methods and thus the different functionalities exposed by the class (or package). As for the contribution at the number of forks, we use $AD(j)$, which refers to the ratio of documented public methods, and $NM(j)$, which refers to the number of methods of the j-th class (or package), and therefore can be used as a measure of its functionalities. Note that the provided functionalities pose a stronger criterion for determining the reusability score of a source code component compared to the documentation ratio, which contributes more as the number of methods approaches to zero. Lastly, as seen in equation (3), the logarithmic scale is applied as a smoothing factor for the diversity in the number of classes and packages among different repositories. This smoothing factor is crucial, since this diversity does not reflect the true quality difference among the repositories.

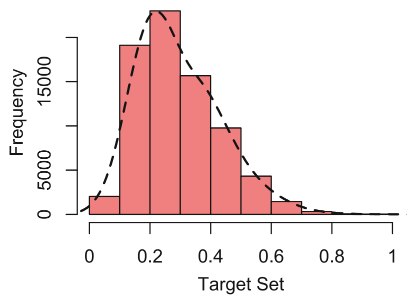
Figure 1 illustrates the distribution of the quality score (target set) for the benchmark dataset classes and packages. Figure 1(a) refers to classes, while Fig. 1(b) refers to packages. The majority of instances for both distributions are accumulated in the interval [0.1, 0.5] and their frequency is decreasing as the score reaches 1. This is expected, since the distributions of the ratings (stars or forks) provided by developers typically exhibit few extreme values.

4 System Design

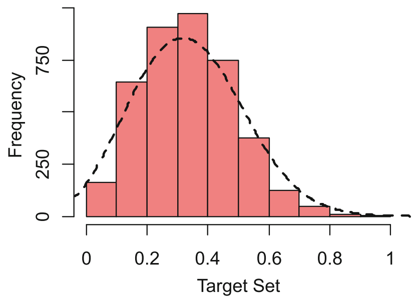
In this section we design our system for quality estimation based on static analysis metrics. We split the dataset of the previous section into two sets, one for training and one for testing. The training set includes 90 repositories with 91531 classes distributed within 6632 packages and the test set includes 12 repositories with 8702 classes distributed within 738 packages. For the training, we used all available static analysis metrics except for those used for constructing the target variable. In specific, AD, NPM, NM, and NCL were used only for the preprocessing stage and then excluded from the models training to avoid skewing the results. In addition, any components with missing metric values are removed (e.g. empty class files or package files containing no classes); hence the updated training set contains 5599 packages with 88180 class files and the updated test set contains 556 packages with 7998 class files.

4.1 System Overview

Our system is shown in Fig. 3. The input is given in the form of static analysis metrics, while the stars and forks of the GitHub repositories are required only for the training of the system. As a result, the developer can provide a set of classes or packages (or a full project), and receive a comprehensible quality analysis as output. Our methodology involves three stages: the preprocessing stage, the metrics selection stage, and the model estimation stage. During preprocessing, the target set is constructed using the analysis of Sect. 3, and the dataset is cleaned



(a)



(b)

Fig. 1. Distribution of the computed quality score at (a) class and (b) package level.

of duplicates and outliers. Metrics selection determines which metrics will be used for each metric category, and model estimation involves training 5 models, one for each category. The stages are analyzed in the following paragraphs.

4.2 Data Preprocessing

The preprocessing stage is used to eliminate potential outliers from the dataset and thus make sure that the models are trained as effectively as possible. To do so, we developed a one-class classifier for each level (class/package) using *Support Vector Machines (SVM)* and trained it using metrics that were selected by means of *Principal Feature Analysis (PFA)*.

At first, the dataset is given as input in two PFA models which refer to classes and packages, respectively. Each model performs *Principal Component Analysis (PCA)* to extract the most informative *principal components (PCs)* from all metrics applicable at each level. In the case of classes, we have 54 metrics, while in the case of packages, we have 68. According to our methodology, we keep the first 12 principal components, preserving 82.8% of the information in the case

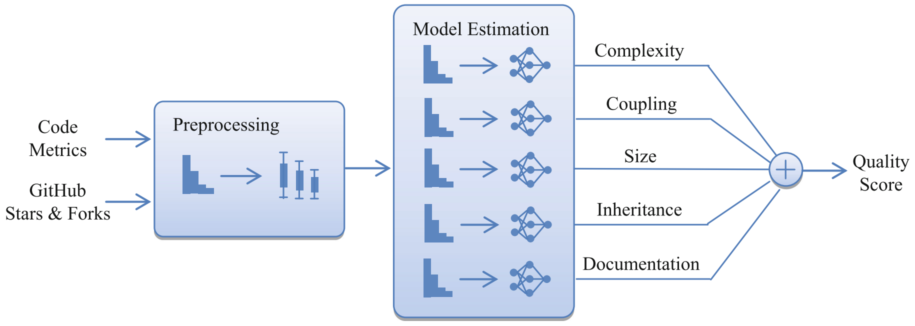


Fig. 2. Overview of the quality estimation methodology [5].

of classes and 82.91% in the case of packages. Figure 3 depicts the percentage of variance for each principal component. Figure 3(a) refers to class level, while Fig. 3(b) refers to package level. We follow a methodology similar to that of [13] in order to select the features that shall be kept. The transformation matrix generated by each PCA includes values for the participation of each metric in each principal component.

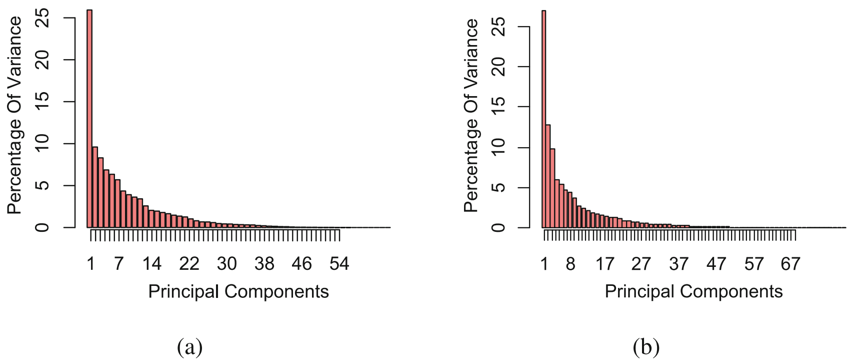


Fig. 3. Variance of principal components at (a) class and (b) package level.

We first cluster this matrix using hierarchical clustering and then select a metric from each cluster. Given that different metrics may have similar trends (e.g. McCabe Complexity with Lines of Code), complete linkage was selected to avoid large heterogeneous clusters. The dendrograms of the clustering for both classes and packages is shown in Fig. 4. Figure 4(a) refers to classes, while Fig. 4(b) refers to packages.

The dendrograms reveal interesting associations among the metrics. The clusters correspond to categories of metrics which are largely similar, such as the metrics of the local class attributes, which include their number (NLA), the number of the public ones (NLPA), and the respective totals (TNLPA and TNLA) that refer to all classes in the file. In both class and package levels, our clustering reveals that keeping one of these metrics results in minimum information loss. Thus, in this case we keep only TNLA. The selection of the kept metric from each cluster in both cases (in red in Fig. 4) was performed by manual examination to end up with a metrics set that conforms to the current state-of-the-practice. An alternative would be to select the metric which is closest to a centroid computed as the Euclidean mean of the cluster metrics.

After having selected the most representative metrics for each case, the next step is to remove any outliers. Towards this direction, we use two SVM one-class classifiers for this task, each applicable at a different level. The classifiers use a radial basis function (RBF) kernel, with *gamma* and *nu* set to 0.01 and 0.1 respectively, and the training error tolerance is set to 0.01. Given that our dataset contains popular high quality source code, outliers in our case are actually low

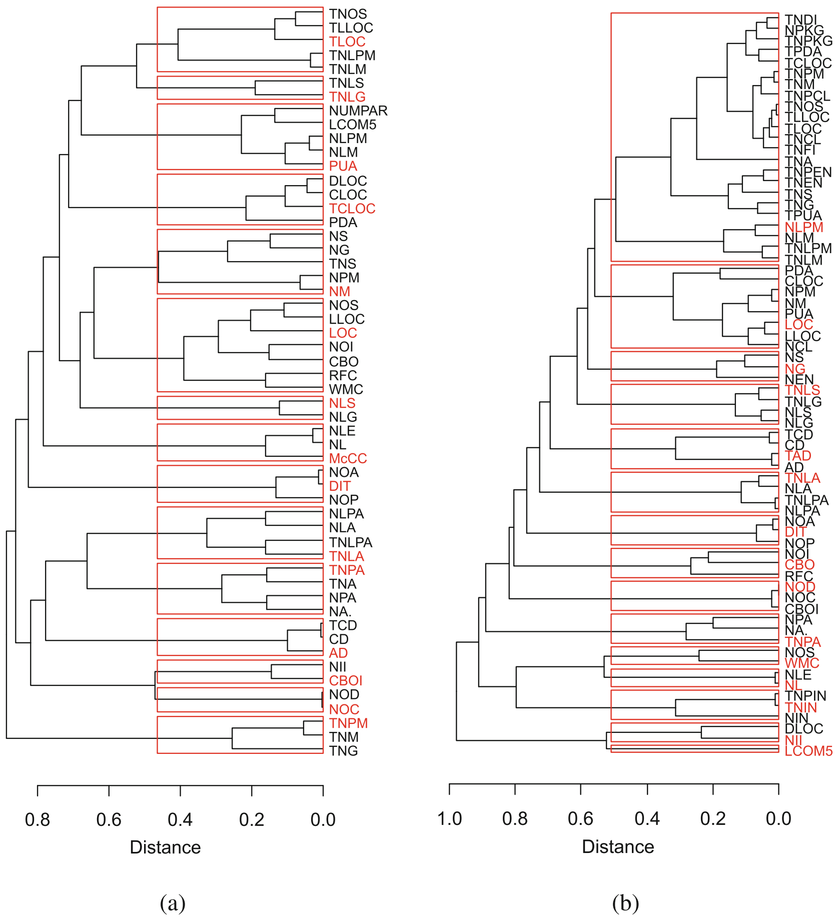


Fig. 4. Dendrogram of metrics clustering at (a) class and (b) package level. (Color figure online)

quality classes or packages. These are discarded since the models of Fig. 2 are trained on high quality source code. As an indicative assessment of our classifier, we use the code violations data described in Sect. 3.

In total, the one-class classifiers ruled out 8815 classes corresponding to 9.99% of the training set and 559 packages corresponding to 9.98% of the training set. We compare the mean number of violations for these rejected classes/packages and for the classes/packages that were accepted, for 8 categories of violations. The results, which are shown in Table 3, indicate that our classifier successfully rules out low quality source code, as the number of violations for both the rejected classes and packages is clearly higher than that of the accepted.

For instance, the classes rejected by the classifier are typically complex since they each have on average approximately one complexity violation; on the other

Table 3. Mean number of violations of accepted and rejected components.

Violation types	Mean number of violations			
	Classes		Packages	
	Accepted	Rejected	Accepted	Rejected
WarningInfo	18.5276	83.0935	376.3813	4106.3309
Clone	4.3106	20.9365	2.9785	10.7513
Cohesion	0.3225	0.7893	0.2980	0.6556
Complexity	0.0976	1.2456	0.0907	0.9320
Coupling	0.1767	1.5702	0.2350	1.2486
Documentation	12.5367	49.9751	13.9128	37.2039
Inheritance	0.0697	0.4696	0.0439	0.2280
Size	1.0134	8.1069	1.2812	5.6296

hand, the number of complexity violations for the accepted classes is minimal. Furthermore, on average each rejected class has more than 8 size violations (e.g. large method bodies), whereas accepted classes have approximately 1.

4.3 Models Preprocessing

Before model construction, we use PFA to select the most important metrics for each of the five metric categories: complexity metrics, coupling metrics, size metrics, inheritance metrics, and documentation metrics. As opposed to data preprocessing, PFA is now used separately per category of metrics. We also perform discretization on the float variables (TCD, NUMPAR, McCC) and on the target variable and remove any duplicates in order to reduce the size of the dataset and thus improve the training of the models.

Analysis at Class Level

Complexity Model. The dataset has four complexity metrics: NL, NLE, WMC, and McCC. Using PCA and keeping the first 2 PCs (84.49% of the information), the features are split in 3 clusters. Figure 5(a) shows the correlation of the metrics with the first two PCs, with the selected metrics (NL, WMC, and McCC) in red.

Coupling Model. The coupling metrics are CBO, CBOI, NOI, NII, and RFC. By keeping the first 2 PCs (84.95% of the information), we were able to select three of them, i.e. CBO, NII, and RFC, so as to train the ANN. Figure 5(b) shows the metrics in the first two PCs, with the selected metrics in red.

Documentation Model. The dataset includes five documentation metrics (CD, CLOC, DLOC, TCLOC, TCD), out of which DLOC, TCLOC, and TCD were found to effectively cover almost all valuable information (2 principal components

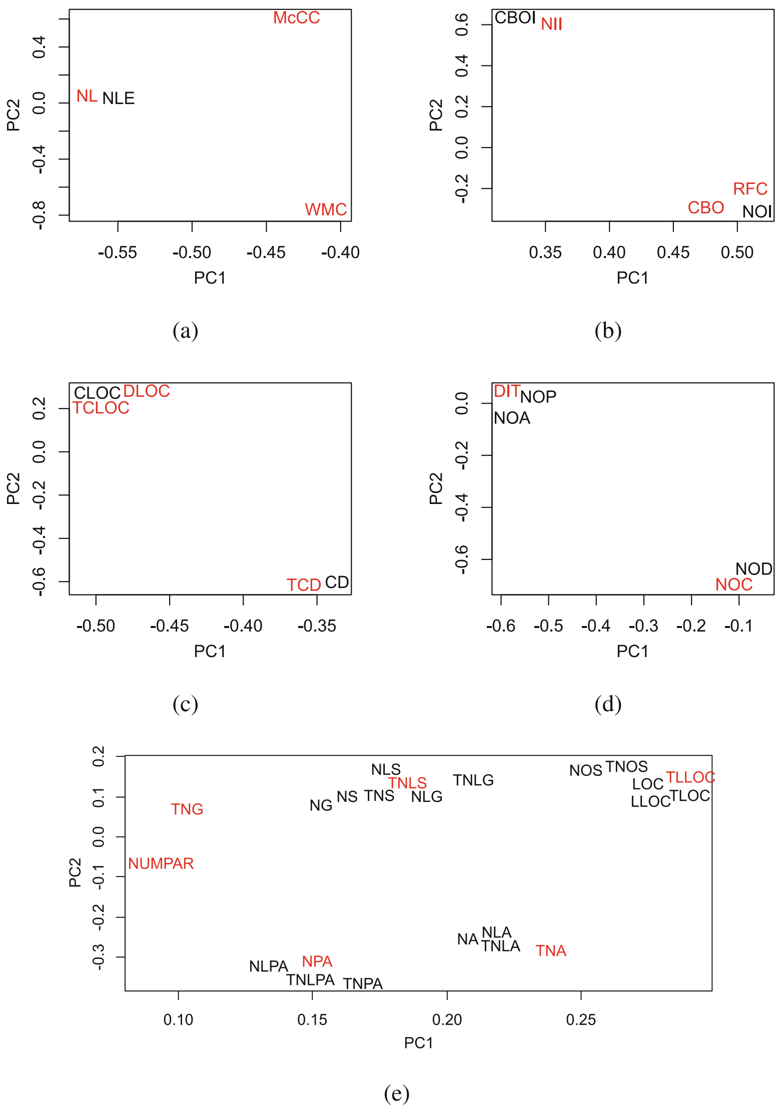


Fig. 5. Visualization of the top 2 PCs at class level for (a) complexity, (b) coupling, (c) documentation, (d) inheritance and (e) size property [5]. (Color figure online)

with 98.73% of the information). Figure 5(c) depicts the correlation of the metrics with the kept components, with the selected metrics in red.

Inheritance Model. For the inheritance metrics (DIT, NOA, NOC, NOD, NOP), the PFA resulted in 2 PCs and two metrics, DIT and NOC, for 96.59% of the information. Figure 5(d) shows the correlation of the metrics with the PCs, with the selected metrics in red.

Size Model. The PCA for the size metrics indicated that almost all information, 83.65%, is represented by the first 6 PCs, while the first 2 (i.e. 53.80% of the variance) are visualized in Fig. 5(e). Upon clustering, we select NPA, TLLOC, TNA, TNG, TNLS, and NUMPAR in order to cover most information.

Analysis at Package Level

Complexity Model. The dataset has three complexity metrics: WMC, NL and NLA. After using PCA and keeping the first two PCs (98.53% of the information), the metrics are split in 2 clusters. Figure 6(a) depicts the correlation of the metrics with the PCs, with the selected metrics (NL and WMC) in red.

Coupling Model. Regarding the coupling metrics, which for the dataset are CBO, CBOI, NOI, NII, and RFC, three of them were found to effectively cover most of the valuable information. In this case the first three principal components were kept, which correspond to 90.29% of the information. The correlation of each metric with the first two PCs is shown in Fig. 6(b), with the selected metrics (CBOI, NII and RFC) in red.

Documentation Model. For the documentation model, upon using PCA and keeping the first two PCs (86.13% of the information), we split the metrics in 3 clusters and keep TCD, DLOC and TCLOC as the most representative metrics. Figure 6(c) shows the correlation of the metrics with the PCs, with the selected metrics in red.

Inheritance Model. The inheritance dataset initially consists of DIT, NOA, NOC, NOD and NOP. By applying PCA, 2 PCs were kept (93.06% of the information). The process of selecting metrics resulted in 2 clusters, of which NOC and DIT were selected as the Fig. 6(d) depicts.

Size Model. The PCA for this category indicated that the 83.57% of the information is successfully represented by the 6 first principal components. Thus, as Fig. 6(e) visualizes, NG, TNIN, TLLOC, NPA, TNLA and TNLS were selected out of 33 size metrics of the original dataset.

4.4 Models Validation

We train five Artificial Neural Network (ANN) models for each level (class and package), each one of them corresponding to one of the five metric properties. All networks have one input, one hidden, and one output layer, while the number of nodes for each layer and each network is shown in Table 4.

10-fold cross-validation was performed to assess the effectiveness of the selected architectures. The validation error for each of the 10 folds and for each of the five models is shown in Fig. 7.

Upon validating the architectures that were selected for our neural networks, in the following paragraphs, we describe our methodology for training our models.

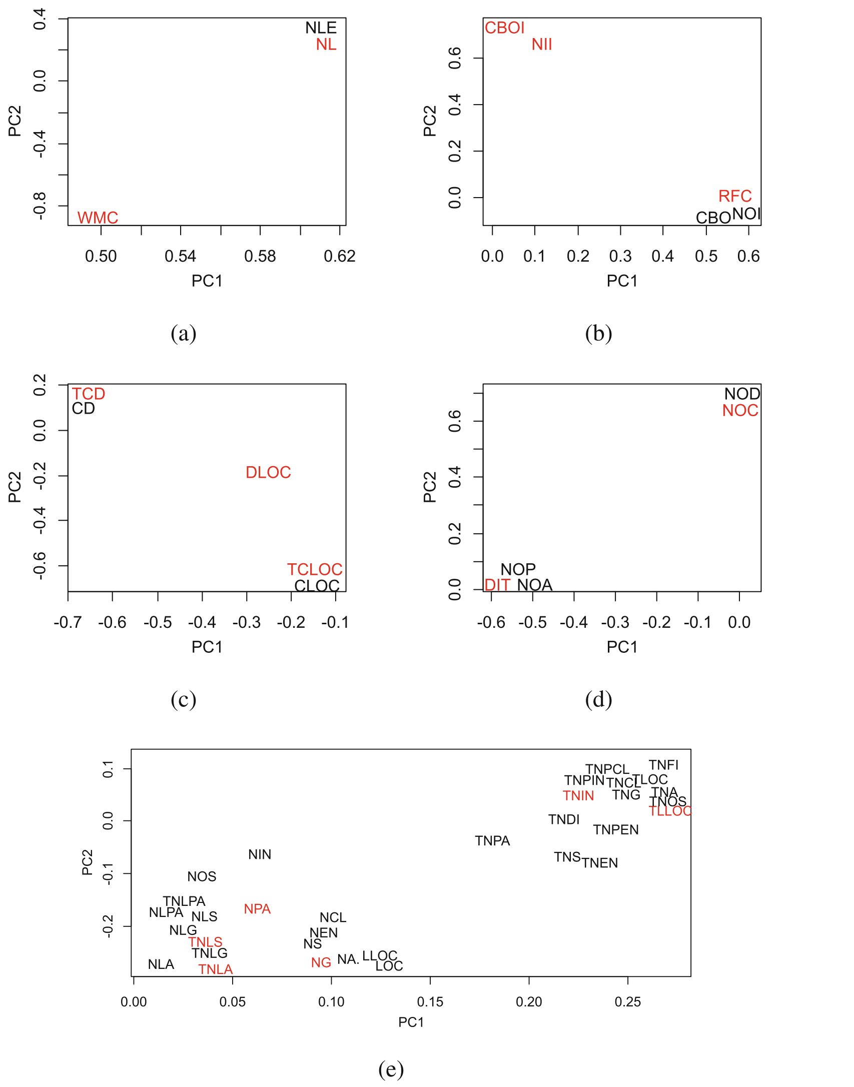


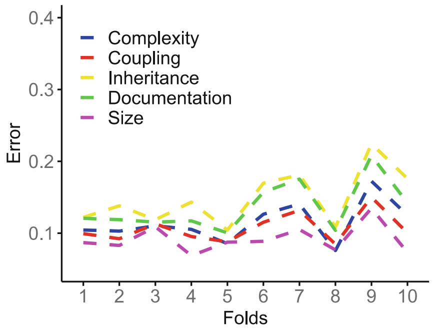
Fig. 6. Visualization of the top 2 PCs at package level for (a) complexity, (b) coupling, (c) documentation, (d) inheritance and (e) size property. (Color figure online)

4.5 Models Construction

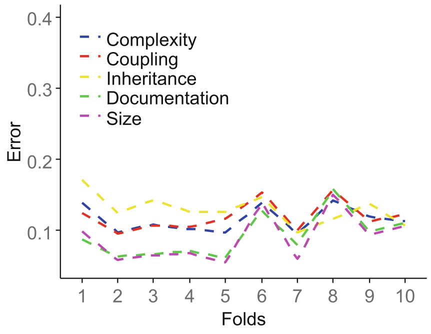
The model construction stage involves the training of five ANN models for each level (class and package) using the architectures defined in the previous subsection. For each level, every model provides a quality score regarding a specific metrics category, and all the scores are then aggregated to provide a final quality score for a given component. Although simply using the mean of the metrics is reasonable, we use weights to effectively cover the requirements of each

Table 4. Neural network architecture for each metrics category.

Metrics category	Class		Package	
	Input nodes	Hidden nodes	Input nodes	Hidden nodes
Complexity	3	1	2	2
Coupling	3	2	3	3
Documentation	3	2	3	3
Inheritance	2	2	2	2
Size	6	4	6	4



(a)



(b)

Fig. 7. 10-fold cross-validation error for the 5 ANNs referring to (a) class level and (b) package level.

individual developer. For instance, a developer may be more inclined towards finding a well-documented component even if it is somewhat complex. In this case, the weights of complexity and documentation could be adapted accordingly.

The default weight values for the models applicable at each level are set according to the correlations between the metrics of each ANN and the respective target score. Thus, for the complexity score, we first compute the correlation of each metric with the target score (as defined in Sect. 3), and then calculate the mean of the absolutes of these values. The weights for the other categories are computed accordingly and all values are normalized so that their sum is equal to 1. The computed weights for the models of each level are shown in Table 5, while the final score is calculated by multiplying the individual scores with the respective weights and computing their sum. Class level weights seem to be more evenly distributed than package level weights. Interestingly, package level weights for complexity, coupling, and inheritance are lower than those of documentation and size, possibly owing to the fact that the latter categories include only metrics computed directly at package level (and not aggregated from class level metrics).

Table 5. Quality score aggregation weights.

Metrics category	Aggregation weights	
	Class level	Package level
Complexity	0.207	0.192
Coupling	0.210	0.148
Documentation	0.197	0.322
Inheritance	0.177	0.043
Size	0.208	0.298

Figure 8 depicts the error distributions for the training and test sets of the aggregated model at both levels (class and package), while the mean error percentages are in Table 6.

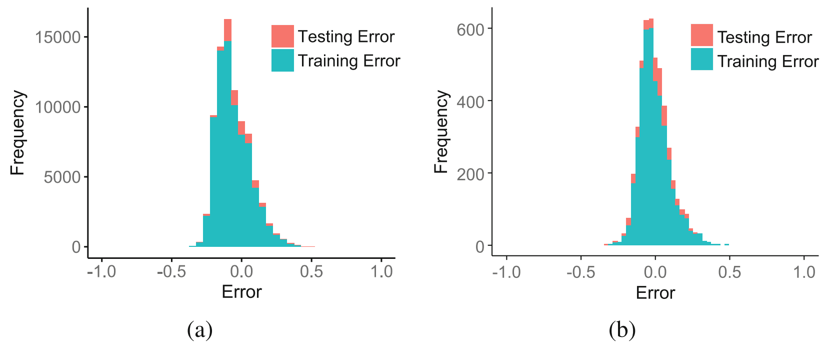


Fig. 8. Error histograms for the aggregated model at (a) class and (b) package level.

The ANNs are trained effectively, as their error rates are low and concentrate mostly around 0. The differences in the distributions between the training and test sets are also minimal, indicating that both models avoided overfitting.

5 Evaluation

5.1 One-Class Classifier Evaluation

Each one-class classifier (one for each level) is evaluated on the test set using the code violations data described in Sect. 3. Regarding the class level, our classifier ruled out 1594 classes corresponding to 19.93% of the classes, while for the package level, our classifier ruled out 89 packages corresponding to 16% of the packages. The mean number of violations for the rejected and the accepted classes and packages are shown in Table 7, for all the 8 categories of violations.

Table 6. Mean error percentages of the ANN models.

Metrics category	Error at class level		Error at package level	
	Training	Testing	Training	Testing
Complexity	10.44%	9.55%	11.20%	9.99%
Coupling	10.13%	8.73%	10.81%	10.08%
Documentation	11.13%	10.22%	7.62%	9.52%
Inheritance	13.62%	12.04%	12.15%	10.98%
Size	9.15%	8.73%	7.15%	9.21%
Final	11.35%	8.79%	7.86%	8.43%

Table 7. Number of violations of accepted and rejected components.

Violation types	Mean number of violations			
	Classes		Packages	
	Rejected	Accepted	Rejected	Accepted
WarningInfo	57.6481	17.4574	1278.4831	312.3640
Clone	18.8338	4.1953	13.2359	2.4935
Cohesion	0.5922	0.3003	0.4831	0.2987
Complexity	1.5772	0.0963	1.3033	0.0985
Coupling	1.4737	0.2099	0.9494	0.2109
Documentation	26.2083	11.4102	23.9213	12.5620
Inheritance	1.2516	0.2854	0.5112	0.1113
Size	7.7114	0.9599	6.0505	1.2751

5.2 Quality Estimation Evaluation

Class Level. Although the error rates of our system are quite low (see Fig. 8), we also have to assess whether its estimations are reasonable from a quality perspective. This type of evaluation requires examining the metric values, and studying their influence on the quality scores. To do so, we use a project as a case study. The selected project, MPAndroidChart, was chosen at random as the results are actually similar for all projects. For each of the 195 class files of the project, we applied our methodology to construct the five scores corresponding to the source code properties and aggregated them for the final quality score.

We use Parallel Coordinates Plots combined with Boxplots to examine how quality scores are affected by the static analysis metrics (Figs. 9(a)–(f)). For each category, we first calculate the quartiles for the score and construct the Boxplot. After that, we split the data instances (metrics values) in four intervals according to their quality score: $[min, q1)$, $[q1, med)$, $[med, q3)$, $[q3, max]$, where min and max are the minimum and maximum score values, med is the median value, and $q1$ and $q3$ are the first and third quartiles, respectively. Each line represents the mean values of the metrics for a specific interval. For example, the blue line

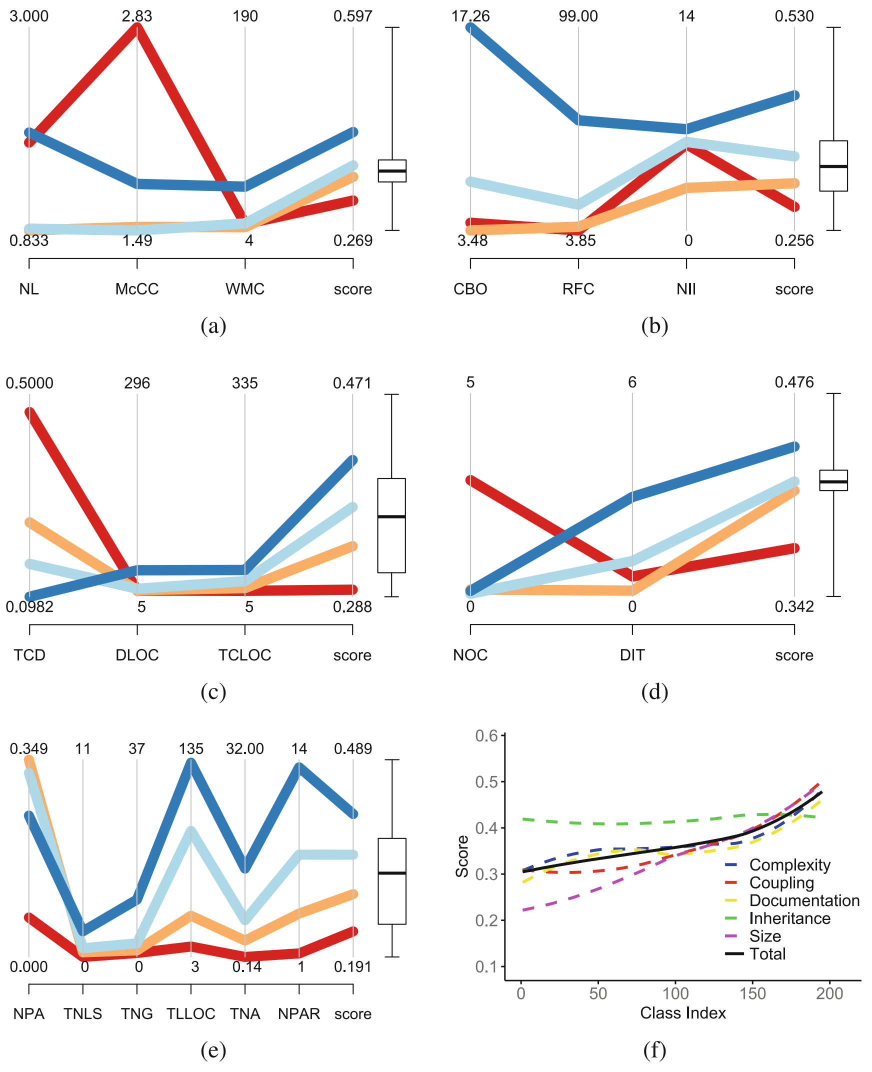


Fig. 9. Parallel Coordinates Plots at class level for the score generated from (a) the complexity model, (b) the coupling model, (c) the documentation model, (d) the inheritance model, (e) the size model, and (f) plot showing the score aggregation [5]. (Color figure online)

refers to instances with scores in the $[q3, max]$ interval. The line is constructed by the mean values of the metrics NL, $McCC$, WMC and the mean quality score in this interval, which are 1.88, 1.79, 44.08, and 0.43 respectively. The red, orange, and cyan lines are constructed similarly using the instances with scores in the $[min, q1)$, $[q1, med)$, and $[med, q3)$ intervals, respectively.

Figure 9(a) refers to the complexity model. This plot results in the identification of two dominant trends that influence the score. At first, $McCC$ appears to be crucial for the final score. High values of the metric result in low score, while low ones lead to high score. This is expected since complex classes are prone to containing bugs and overall imply low quality code. Secondly, the metrics WMC and NL do not seem to correlate with the score individually; however they affect it when combined. Low WMC values combined with high NL values result in low quality scores, which is also quite rational given that more complex classes with multiple nested levels are highly probable to exhibit low quality.

Figures 9(b) and (c) refer to the coupling and the documentation models, respectively. Concerning coupling, the dominant metric for determining the score appears to be RFC. High values denote that the classes include many different methods and thus many different functionalities, resulting in high quality score. As for the documentation model, the plot indicates that classes with high comment density (TCD) and low number of documentation lines ($DLOC$) are given a low quality score. This is expected as this combination probably denotes that the class does not follow the Java documentation guidelines, i.e. it uses comments instead of Javadoc.

Figures 9(d) and (e) refer to the inheritance and size models, respectively. DIT appears to greatly influence the score generated by the inheritance model, as its values are proportional to those of the score. This is expected as higher values indicate that the class is more independent as it relies mostly on its ancestors, and thus it is more reusable. Although higher DIT values may lead to increased complexity, the values in this case are within acceptable levels, thus the score is not negatively affected.

As for the size model, the quality score appears to be mainly influenced by the values of $TLLOC$, TNA and $NUMPAR$. These metrics reflect the amount of valuable information included in the class by measuring the lines of code and the number of attributes and parameters. Classes with moderate size and many attributes or parameters seem to receive high quality scores. This is expected as attributes/parameters usually correspond to different functionalities. Additionally, a moderately sized class is common to contain considerable amount of valuable information while not being very complex.

Finally, Fig. 9(f) illustrates how the individual quality scores (dashed lines) are aggregated into one final score (solid line), which represents the quality degree of the class as perceived by developers. The class indexes (project files) are sorted in descending order of quality score. The results for each score illustrate several interesting aspects of the project. For instance, it seems that the classes exhibit similar inheritance behavior throughout the project. On the other hand, the size quality score is diverse, as the project has classes with various size characteristics (e.g. small or large number of methods), and thus their score may be affected accordingly. Finally, the trends of the individual scores are in line with the final score, while their variance gradually decreases as the final score increases. This is expected as a class is typically of high quality if it exhibits acceptable metric values in several categories.

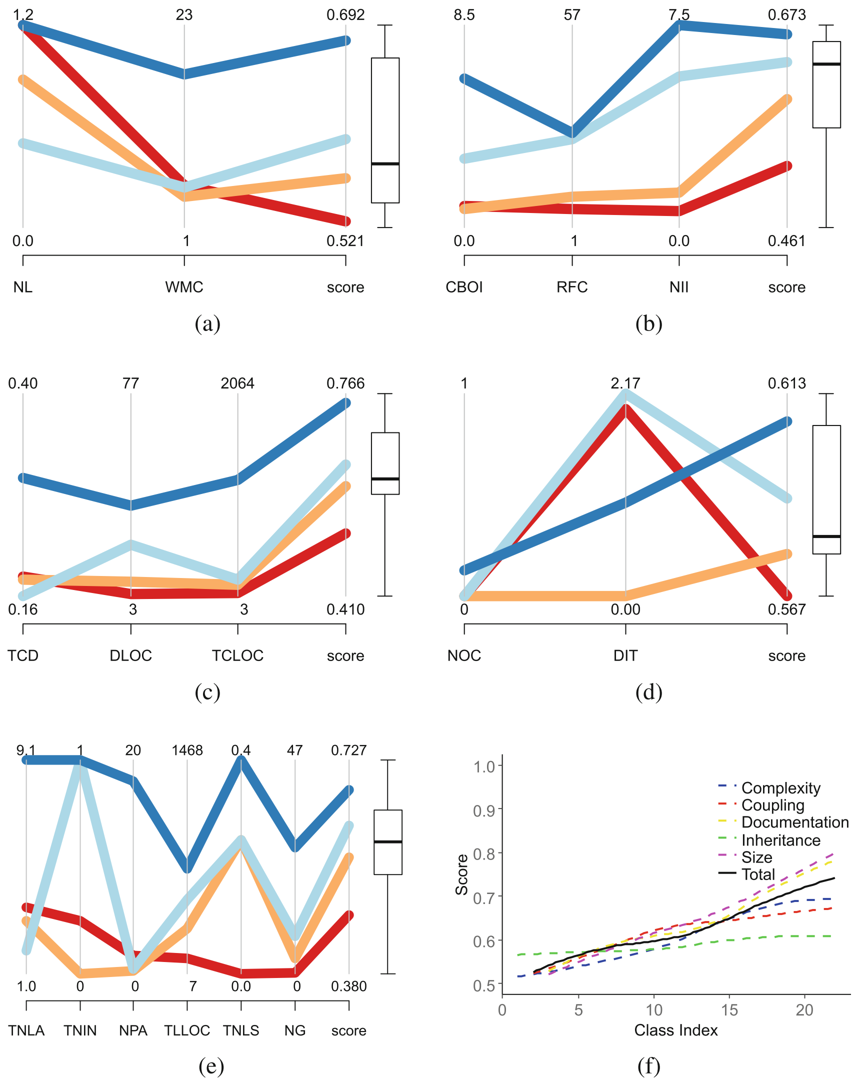


Fig. 10. Parallel Coordinates Plots at package level for the score generated from (a) the complexity model, (b) the coupling model, (c) the documentation model, (d) the inheritance model, (e) the size model, and (f) plot showing the score aggregation.

Package Level. Following the same strategy as in the case of classes, we constructed Parallel Coordinates Plots combined with Boxplots towards examining the influence of the values of the static analysis metrics on the quality score. Figure 10 depicts the plots for each of the five source code properties under evaluation and the aggregated plot of the final quality score.

At this point, it is worth noticing that only in the cases of size and documentation, the values of the static analysis metrics originate from the packages themselves, while for the other three models the values of the static analysis metrics originate from classes. As a result, the behaviors extracted in the cases of size and documentation are considered more accurate which originates from the fact that they do not accumulate noise due to aggregations. As already noted in Subsect. 3.1, the median was used as an aggregation mechanism, which is arguably an efficient measure as it is at least not easily influenced by extreme metrics' values.

Figure 10(a) refers to the complexity model. As it can be seen from the diagram, the outcome of the complexity score appears to be highly influenced by the values of WMC metric. High WMC values result in high score while lower values appear to have the opposite impact. Although this is not expected as higher complexity generally is interpreted as an negative characteristic, in this case, given the intervals of the complexity-related metrics, we can see that the project under evaluation appears to exhibit very low complexity. This is reflected in the intervals of both NL and WMC which are [0, 1.2] and [0, 23], respectively. Consequently, the extracted behaviour regarding the influence of WMC in the outcome of the final score can be considered logical as extremely low values of WMC (close to zero) indicate absence of valuable information and thus the score is expected to be low.

Figures 10(b) and (c) refer to the coupling and the documentation model, respectively. In the case of coupling, it is obvious that the values of the NII (Number of Incoming Invocations) metric appear to highly influence the outcome of the final score. High NII values result in high score, while low values appear to have a negative impact. This is expected as NII metric reflects the significance of a given package due to the fact that it measures the number of other components that call its functions. In addition, we can see that high values of CBOI (Coupling Between Objects Inverse) metric result in high coupling score which is totally expected as CBOI reflects how decoupled is a given component. As for the documentation model, it is obvious that the Total Comments Density (TCD) metric appears to influence the outcome of the final score. Moderate values (around 20%) appear to result in high scores which is logical considering the fact that those packages appear to have one line of comment for every five lines of code.

Figures 10(d) and (e) refer to the inheritance and the size model, respectively. As for the inheritance model, DIT metric values appear to greatly influence the generated score in a proportional manner. This is expected as higher DIT values indicate that a component is more independent as it relies mostly on its ancestors, and thus it is more reusable. It is worth noticing that although higher DIT values may lead to increased complexity, the values in this case are within acceptable levels, thus the score is not negatively affected. As for the size model, the packages that appear to have normal size as reflected in the values of TLLOC (Total Logical Lines Of Code) metric receive high score. On the other hand, the ones that appear to contain little information receive low score, as expected.

Finally, Fig. 10(f) illustrates how the individual quality scores (dashed lines) are aggregated into one final score (solid line), which represents the quality degree of the package as perceived by developers. The package indexes are sorted in descending order of quality score. Similar to the case of classes, the trends of the individual scores are in line with the final score. The score that originates from the inheritance model exhibits the highest deviation from the final score, while the lowest deviation is the one of the scores originating from the size and the documentation models. This is expected as those two properties include metrics that are applicable directly at package level.

5.3 Example Quality Estimation

Further assessing the validity of our system, for each category we manually examine the values of the static analysis metrics of 20 sample components (10 classes and 10 packages) that received both high and low quality scores regarding each one of the five source code properties, respectively. The scores for these classes and packages are shown in Table 8. Note that the presented static analysis metrics refer to different classes and packages for each category. For the complexity model, the class that received low score appears to be much more complex than the one that received high score. This is reflected in the values of McCC and NL, as the low-scored class includes more complex methods (8.5 versus 2.3), while it also has more nesting levels (28 versus 4). The same applies for the packages that received high and low scores, respectively.

For the coupling model, the high-quality class has significantly higher NII and RFC values when compared to those of the low-quality class. This difference in the number of exposed functionalities is reflected in the quality score. The same applies for the inheritance model, where the class that received high score is a lot more independent (higher DIT) and thus reusable than the class with the low score. The same conclusions can be derived for the case of packages where it is worth noticing that the difference between the values of the coupling-related metrics between the high-scored and the low-scored package are smaller. This is a result of the fact that the described coupling metrics are only applicable at class level.

As for the inheritance score, it is obvious in both the cases of classes and packages that the higher degree of independence as reflected in the low values of NOC and NOP metrics results into high score. Finally, as for the documentation and size models, in both cases the low-quality components (both classes and packages) appear to have no valuable information. In the first case, this absence is obvious from the extreme value of comments density (TCD) combined with the minimal documentation (TCLOC). In the second case, the low-quality class and package contain only 10 and 40 logical lines of code (TLLOC), respectively, which indicates that they are of almost no value for the developers. On the other hand, the high-quality components seem to have more reasonable metrics values.

Table 8. Static analysis metrics per property for 20 components (10 classes and 10 packages) with different quality scores.

Metrics		Classes		Packages	
Category	Name	High score (80–90%)	Low score (10–15%)	High score (80–90%)	Low score (10–15%)
Complexity	McCC	2.3	8.5	–	–
	WMC	273	51	12	6
	NL	4	28	2	4
Coupling	NII	88	0	21.5	4
	RFC	65	7	30	8
	CBO	5	35	3	14
Documentation	TCD	0.3	0.8	0.35	0.82
	DLOC	917	2	372	7
	TCLOC	1,019	19	2654	24
Inheritance	DIT	8	0	3	0
	NOC	1	16	1	9
	NOP	2	14	2	8
Size	NUMPAR	27	3	–	–
	NCL	–	–	9	1
	TNA	69	0	65	2
	NPA	36	0	29	0
	TLLOC	189	10	1214	40
	TNLS	13	2	78	4
	NM	9	1	98	1

5.4 Threats to Validity

The threats to the validity of our approach and our evaluation involve both its applicability to software projects and its usage by the developers. Concerning applicability, the dataset used is quite diverse; hence our methodology can be seamlessly applied to any software project for which static analysis metrics can be extracted. Concerning expected usage, developers would harness the quality estimation capabilities of our approach in order to assess the quality of their own or third-party software projects before (re)using them in their source code. Future work on this aspect may involve integrating our approach in a system for software component reuse, either as an online component search engine or as an IDE plugin.

6 Conclusions

Given the late adoption of a component-based software engineering paradigm, the need for estimating the quality of software components before reusing them (or before publishing one's components) is more eminent than ever. Although previous work on the area of designing quality estimation systems is broad, there is usually some reliance on expert help for model construction, which in turn may lead to context-dependent and subjective results. In this work, we employed information about the popularity of source code components to model their quality as perceived by developers, an idea originating from [15] that was found to be effective for estimating the quality of software classes [5].

We have proposed a component-based quality estimation approach, which we construct and evaluate using a dataset of source code components, at class and package level. Upon removing outliers using a one-class classifier, we apply Principal Feature Analysis techniques to effectively determine the most informative metrics lying in five categories: complexity, coupling, documentation, inheritance, and size metrics. The metrics are subsequently given to five neural networks that output quality scores. Our evaluation indicates that our system can be effective for estimating the quality of software components as well as for providing a comprehensive analysis on the aforementioned five source code quality axes.

Future work lies in several directions. At first, the design of our target variable can be further investigated for different scenarios and different application scopes. In addition, various feature selection techniques and models can be tested to improve on current results. Finally, we could assess the effectiveness of our methodology by means of a user study, and thus further validate our findings.

References

1. Alves, T.L., Ypma, C., Visser, J.: Deriving metric thresholds from benchmark data. In: IEEE International Conference on Software Maintenance (ICSM), pp. 1–10. IEEE (2010)
2. Cai, T., Lyu, M.R., Wong, K.F., Wong, M.: ComPARE: a generic quality assessment environment for component-based software systems. In: proceedings of the 2001 International Symposium on Information Systems and Engineering (2001)
3. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. **20**(6), 476–493 (1994)
4. Diamantopoulos, T., Thomopoulos, K., Symeonidis, A.: QualBoa: reusability-aware recommendations of source code components. In: IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pp. 488–491. IEEE (2016)
5. Dimaridou, V., Kyprianidis, A.C., Papamichail, M., Diamantopoulos, T., Symeonidis, A.: Towards modeling the user-perceived quality of source code using static analysis metrics. In: 12th International Conference on Software Technologies (ICSOFT), Madrid, Spain, pp. 73–84 (2017)
6. Ferreira, K.A., Bigonha, M.A., Bigonha, R.S., Mendes, L.F., Almeida, H.C.: Identifying thresholds for object-oriented software metrics. J. Syst. Softw. **85**(2), 244–257 (2012)

7. Foucault, M., Palyart, M., Falleri, J.R., Blanc, X.: Computing contextual metric thresholds. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, pp. 1120–1125. ACM (2014)
8. Hegedűs, P., Bakota, T., Ladányi, G., Faragó, C., Ferenc, R.: A drill-down approach for measuring maintainability at source code element level. Electron. Commun. EASST **60** (2013)
9. Heitlager, I., Kuipers, T., Visser, J.: A practical model for measuring maintainability. In: 6th International Conference on the Quality of Information and Communications Technology, QUATIC 2007, pp. 30–39. IEEE (2007)
10. ISO/IEC 25010:2011 (2011). https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en. Accessed Nov 2017
11. Kanellopoulos, Y., Antonellis, P., Antoniou, D., Makris, C., Theodoridis, E., Tjortjis, C., Tsirakis, N.: Code quality evaluation methodology using the ISO/IEC 9126 standard. Int. J. Softw. Eng. Appl. **1**(3), 17–36 (2010)
12. Le Goues, C., Weimer, W.: Measuring code quality to improve specification mining. IEEE Trans. Softw. Eng. **38**(1), 175–190 (2012)
13. Lu, Y., Cohen, I., Zhou, X.S., Tian, Q.: Feature selection using principal feature analysis. In: Proceedings of the 15th ACM International Conference on Multimedia, pp. 301–304. ACM (2007)
14. Miguel, J.P., Mauricio, D., Rodríguez, G.: A review of software quality models for the evaluation of software products. arXiv preprint arXiv:1412.2977 (2014)
15. Papamichail, M., Diamantopoulos, T., Symeonidis, A.: User-perceived source code quality estimation based on static analysis metrics. In: IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 100–107. IEEE (2016)
16. Pfleeger, S.L., Atlee, J.M.: Software Engineering: Theory and Practice. Pearson Education India, Delhi (1998)
17. Pfleeger, S., Kitchenham, B.: Software quality: the elusive target. IEEE Softw. **13**, 12–21 (1996)
18. Samoladas, I., Gousios, G., Spinellis, D., Stamelos, I.: The SQO-OSS quality model: measurement based open source software evaluation. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (eds.) OSS 2008. ITIFIP, vol. 275, pp. 237–248. Springer, Boston, MA (2008). https://doi.org/10.1007/978-0-387-09684-1_19
19. Schmidt, C.: Agile Software Development Teams. Progress in IS. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-26057-0
20. Shatnawi, R., Li, W., Swain, J., Newman, T.: Finding software metrics threshold values using ROC curves. J. Softw.: Evol. Process **22**(1), 1–16 (2010)
21. SourceMeter static analysis tool (2017). https://www.sourcemeter.com/. Accessed Nov 2017
22. Taibi, F.: Empirical analysis of the reusability of object-oriented program code in open-source software. Int. J. Comput. Inf. Syst. Control Eng. **8**(1), 114–120 (2014)
23. Washizaki, H., Namiki, R., Fukuoka, T., Harada, Y., Watanabe, H.: A framework for measuring and evaluating program source code quality. In: Münch, J., Abrahamsson, P. (eds.) PROFES 2007. LNCS, vol. 4589, pp. 284–299. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73460-4_26
24. Zhong, S., Khoshgoftaar, T.M., Seliya, N.: Unsupervised learning for expert-based software quality estimation. In: HASE, pp. 149–155 (2004)

A Technology for Optimizing the Process of Maintaining Software Up-to-Date

Andrei Panu(✉)

Faculty of Computer Science, Alexandru Ioan Cuza University of Iasi, Iasi, Romania
andrei.panu@info.uaic.ro

Abstract. In this paper we propose a solution for reducing the time needed to make changes in an application in order to support a new version of a software dependency (e.g., library, interpreter). When such an update is available, we do not know if it comes with some changes that can break the execution of the application. This issue is very serious in the case of interpreted languages, because errors appear at runtime. System administrators and software developers are directly affected by this problem. Usually the administrators do not know many details about the applications hosted on their infrastructure, except the necessary execution environment. Thus, when an update is available for a library packaged separately or for an interpreter, they do not know if the applications will run on the new version, being very hard for them to take the decision to do the update. The developers of the application must make an assessment and support the new version, but these tasks are time consuming. Our approach automates this assessment by analyzing the source code and verifying if and how the changes in the new version affect the application. By having such kind of information obtained automatically, it is easier for system administrators to take a decision regarding the update and it is faster for developers to find out which is the impact of the new version.

Keywords: Information extraction · Named entity recognition
Machine learning · Web mining · Software maintenance

1 Introduction

Cloud computing brought some significant changes in software development, deployment, and execution. It has also changed the way we use and interact with software applications, as professionals or as consumers. Nowadays we have increasingly more applications that are accessible from everywhere using the Internet, which do not need to be locally installed, and which are provided as a service (SaaS – Software as a Service). These can be monolithic applications of different sizes or can be composed of multiple services that run "in cloud", in various environments. Even mobile applications that are installed locally use

© Springer International Publishing AG, part of Springer Nature 2018
E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 28–48, 2018.
https://doi.org/10.1007/978-3-319-93641-3_2

backend services hosted on servers. All of them are executed in pre-configured environments. After the applications are initially developed and deployed, some challenges appear regarding their maintenance and the maintenance of their execution environment. For example, when an update is available for the interpreter of a certain language (e.g. PHP, Python, etc.), system administrators face a problem, especially if the update is a major one. One of their major tasks is to maintain the infrastructure for running different applications and keep the systems up to date. When they are faced with updating the interpreter, on which the deployed applications depend, to a new version, they have to answer questions like: will the existing applications run on the new version of the interpreter? Are there any parts of the applications that will not run because of the changes?

These questions can not be answered easily. As they are usually not the developers of the hosted applications or services that rely on the interpreter, they do not know what impact the update will have on their execution. The same situation exists in case of libraries which are packaged and installed independently and on which the applications depend. Given the fact that the system administrators do not have any knowledge about the applications, except the necessary versions of the interpreters/libraries when the applications were developed and deployed, they can only base their decision on assumptions to make the update. One intuitive assumption is that if there is a minor version update for the interpreter or library, everything should be fine, as no major changes in functionality usually occur. In the majority of cases, this holds true, but it is still an assumption, not a certainty. The problem arises when there is a major update and the development team does not plan to update the application to support the new release. The problem is very serious in case of interpreted languages, because errors appear at run-time, only when certain blocks of code get executed, not initially when everything gets compiled, as is the case with compiled languages. Thus, we can have parts of an application that may work and parts that may not work. The system administrators simply will not know if there will be parts of the application that will not execute, they are not the developers, they are not the testers. This problem scales, because a single administrator has multiple applications running on his/her infrastructure. Our proposal offers a solution for this situation, by automatically analyzing and providing information about whether the new targeted version will bring some changes that will break the execution of the hosted applications or not. Thus, the administrators will know upfront if it is safe to do the update.

Nowadays there is also a shift regarding the management of the execution environment, from the dedicated system administrators to the teams developing the applications, through the use of containers like *Docker*. This does not solve the problem, only shifts it to the developers, although it does not mean that administrators do not care about the situation of the environments of the containers running on their infrastructure. Thus, when the development team wants to update the interpreter or some libraries used, it faces the same problem described above. Even though the team knows the internals of the application, it does not know exactly which blocks of code will execute and which will not. The

Table 1. PHP major versions market share.

PHP version	Market share
5	89.7%
7	9.4%
4	0.8%
3	<0.1%

Table 2. PHP 5 minor versions market share.

PHP version	Market share	Initial release	End of life
5.6	32.5%	28 August 2014	31 December 2018
5.4	22.0%	1 March 2012	3 September 2015
5.3	20.8%	30 June 2009	14 August 2014
5.5	15.9%	20 June 2013	21 July 2016
5.2	8.2%	2 November 2006	6 January 2011
5.1	0.5%	24 November 2005	24 August 2006
5.0	<0.1%	13 July 2004	5 September 2005

solution is to make an assessment of the changes that need to be done by analyzing the changelog/migration guide of the interpreter/library. This is a manual procedure which is time consuming. Our solution helps reduce this time.

In order to better visualize the dimension of the problem, let us consider a single example, the case of web applications built with PHP. This is one of the most used server-side programming languages for developing web sites/applications, having a market share of 83% (source accessed on 16 November 2017) [58]. Table 1 presents the market share of the major versions that are used (source [59] accessed on 16 November 2017). As we can see, the most used is PHP 5, which was released on 13 July 2004. The latest version, PHP 7, released on 3 December 2015, is increasingly adopted, but still there are many applications that have not been ported. Having the biggest market share, we will further analyze PHP 5. Table 2 presents the market share of the minor versions that are used (sources [36,60] accessed on 16 November 2017). As we can see, the only version that is still supported is 5.6, but this is used in only a third of existing applications. The second and third most used versions are unsupported since two and three years ago. End of life means that they do not benefit of updates of any kind [37]. Situations like these exist in case of all other languages that are used for developing web applications.

In this paper we propose a novel approach [32,33] that addresses the specific problem of keeping software applications up to date. It gives the administrators an insight regarding the impact of doing the update, and the developers information about the changes to be made, all that in an automatic manner, improving their productivity for such kind of tasks. In the following section we present our approach and give details about how it solves the presented problem. In Sect. 3 we describe a prototype platform that implements our ideas. Section 4 presents some experimental results. Further, in Sect. 5, we present related solutions that address the same problem. Our approach may also be applicable in other fields, use cases which are briefly described in Sect. 6. Finally, Sect. 7 contains future development and research ideas for the evolution of the technology.

2 Our Proposal

The approach that we propose towards solving the aforementioned problem is a technology that is able to automatically scan the source code and verify if the used functionalities are still supported in a targeted version of an interpreter/library. Our solution uses machine learning and natural language processing techniques. It does not depend on the language used to develop the software. As we have stated earlier, the focus is on interpreted languages. This technology is comprised of three parts:

1. A static analysis tool that needs access to the source code, automatically analyzes it, extracts the used functionalities, and queries a knowledge base that helps answer questions like: is a certain functionality supported in the new targeted version? If not, what are the changes that were made?
2. A knowledge base [30] created automatically that contains information about the functionalities supported in every version of the interpreter/library;
3. A platform that extracts specific entities from online or offline manuals, independently of the programming language, and populates and updates the knowledge base.

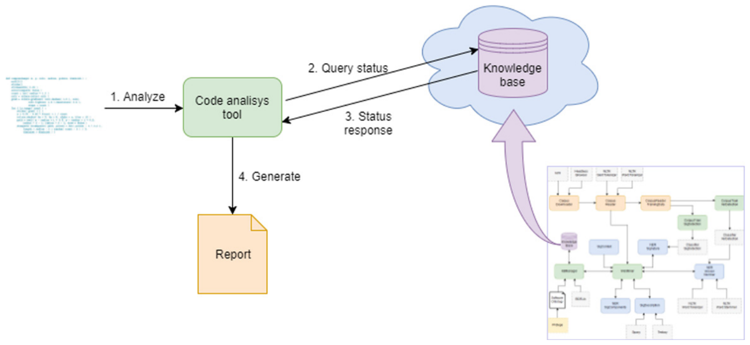


Fig. 1. General workflow. (Source: [32]).

The general workflow of our approach is depicted in Fig. 1 [32]. The key enabler of our technology is the knowledge base. The most important aspect is the contained information and the way it is obtained. For the current version of the platform that populates the knowledge base, which is presented in Sect. 3, the functionalities taken into consideration are the supported functions in all versions of the interpreter/library (the provided APIs). The analysis tool is the part that must have access to the source code and which queries the remote knowledge base to check if there are used functionalities that are not supported anymore (or are marked to be removed in the future) in the targeted version. It generates a report based on the findings. There are many important aspects to be considered for the development of this tool, like:

- how to automatically identify the used libraries;
- how to automatically identify the used version of the libraries;
- how to extract from the code only those functions that are provided by libraries or by the interpreter;
- how to identify where each function is defined (in other words, who provides a certain function).

In this paper, the focus of our work is on the platform that creates and updates the knowledge base. As we have mentioned, the data contained consists of details regarding all the supported functions in targeted interpreters/libraries. For each function, we have different attributes, like its signature's components (the function's name, the number of parameters, the types of the parameters, the order of the parameters), the return type, its short description, its availability (supported, deprecated, obsolete), and the version number of the interpreter/library in which it is supported. The sources of information that are used are online manuals available on the Web or offline ones. The platform described in Sect. 3 is capable to automatically extract the desired data, and does not depend on the content (certain manuals for certain languages/libraries) or on the structure of the web pages. The extraction technology does not have implemented any adapters for specific manuals. It supports any manual for any programming language. The only restriction is regarding the syntax used for writing the functions in the manual. This capability is achieved using machine learning algorithms and different natural language processing techniques.

Being able to automatically obtain such kind of information regarding the unsupported changes offers some major advantages in multiple fields [32,33], like system administration, software maintenance, and even project management. Our solution eases the job of system administrators to maintain their systems up to date. They can now have an insight into what will happen with the execution of the application if an update is made, without waiting for some feedback from the developers. Also, they do not need to know any details regarding the implementation of the application. When the development team is required to ensure maintenance and to update some dependencies, it faces the same difficulty. Although it knows the application very well internally, it does not know exactly which blocks of code will execute and which will not, in case of existing changes of some functions. There are multiple approaches for solving this, like executing unit or functional tests. This represents a very good method to find out if the application works with the new version or what parts of it do not work anymore. The major disadvantage is that there are many situations where these automated tests were not developed or were implemented only for the core of the application. Another approach is to do manual testing, which is a tedious process and prone to human error. Running an automated tool, which checks if deprecated or removed features are used, is another solution, but as we will see in Sect. 5, existing tools are rather limited in scope and coverage. In case of the first two approaches, after identifying the parts of the application that do not work anymore, the developers must make an assessment of the changes that need to be done, by consulting the changelog/migration guide of the dependency's new version. This manual procedure also requires some time. By having the tool that is capable of scanning the code, interrogating our knowledge base and reporting what functions are not supported anymore or suffered some changes, pointing the developers exactly to the blocks of code that need to be modified, a lot of time is earned, by eliminating the manual steps described above. The report also provides details about the differences between the old and the new function(s), easing the job of the developers to make the changes, by not requiring them to manually search for that information. All these aspects will improve the delivery time for the updates. Such a report that can be created automatically, containing all the changes that need to be done, is also very useful in case of project estimation. When there is the intention to support a new version of a software dependency, the developers are required to do an assessment of the changes. As we have seen earlier, this is a tedious and time consuming process. The report has a positive impact on productivity. It can be used by all the persons involved in estimating a project to find out from the start what are the implications of making a certain update (in the context described above), without requiring some developers to make the assessment. It eliminates all that manual labor, thus reducing the time needed to make the estimates.

By employing our solution which needs access to the source code, the administrators and the developers of the applications can have some real privacy concerns. Privacy is a very important issue nowadays, and there are many legal and technological aspects regarding this [2,34]. Privacy is a concept that has a gradual nature, it can not be treated in an "all or nothing" manner. Each software system introduces a different level of risk, so the level of privacy should be "estimated as a measurement of the risk that data can be copied and used outside its context" [2]. This risk regarding unauthorized access to private data should be evaluated according to the information an application works with, considering the impact a data breach could have. In this regard, in the paper cited previously, we have proposed a framework for doing such an evaluation. Our approach and design is in line with principles like Privacy by Design and Privacy by Default [16]. The private data that we work with is represented by the source code. The users of our solution are administrators and developers. A third party can be the provider which hosts the extraction platform and manages the knowledge base. The analysis tool is the only component that poses a privacy risk, because it must run on the users' infrastructure and have access to the private data. This tool extracts only the functions used in the code, which represent public information, they are provided by the interpreter/library, ignoring everything else in the source code. Further, only those functions are transmitted into the requests made to the knowledge base. Effectively no proprietary code is extracted and transmitted. Thus, we consider that this tool must be open source, in order to be easily verifiable that it does not leak the source code. Moreover, if someone intercepts the communication, it will not get sensitive information. Even though the information transmitted over the Internet is formed of publicly available data, the communication channel should still be encrypted, because the targeted version of an interpreter/library is sent and this might be of use for an attacker.

3 System Architecture and Design

In this section we present the architecture of a prototype platform that is capable of automatically accessing the manuals of interpreters/libraries available on the Web or offline, identifying and extracting specific entities, and populating the knowledge base. Its components are designed to be context independent and decoupled. Each component offers its functionality as an independent service. The designed architecture is based on SOA (Service Oriented Architecture) principles. The platform is fully implemented in Python. Figure 2 depicts the system's architecture. It has four main components, *CorpusTrain SigDetection*, *CorpusTrain VerDetection*, *WebMiner*, *OntoManager*, and its functionality is split into two phases, a training phase and an extraction phase.

In the first phase we train two classifiers for the purpose of detecting function signatures and version numbers of a targeted interpreter/library in which the functions are supported. In case of the former, the training data contains manually annotated information taken from PHP and Python online manuals. *CorpusReader* and *CorpusReader TrainingData* are the components used for loading the training data into memory. *CorpusTrain SigDetection* creates the feature sets, splits the data, and trains the classifier, generating *Classifier SigDetection*. In case of the latter, the training data contains manually annotated information taken from PHP, Python, and jQuery online manuals. The same two components previously mentioned are used for loading the training data. *CorpusTrain VerDetection* creates the feature sets, does the splitting, and trains the classifiers, generating *Classifier VerDetection*.

The second phase consist of accessing the manuals, analyzing them, and extracting specific knowledge.orchestratedby *WebMiner*. *CorpusDownloader* is the component that accesses the desired data. Then, the downloaded information is tokenized by *CorpusReader*. The data is sent to *NER SigContext*, which identifies existing signatures and their descriptive contexts. For each signature, *NER SigComponents* extracts the following elements: the function's name, return type, and parameters. Each descriptive context is sent to *NER VersionNumber*, which extracts the version number(s) in which the function is available or in which it was marked as deprecated or removed. Next, the description of

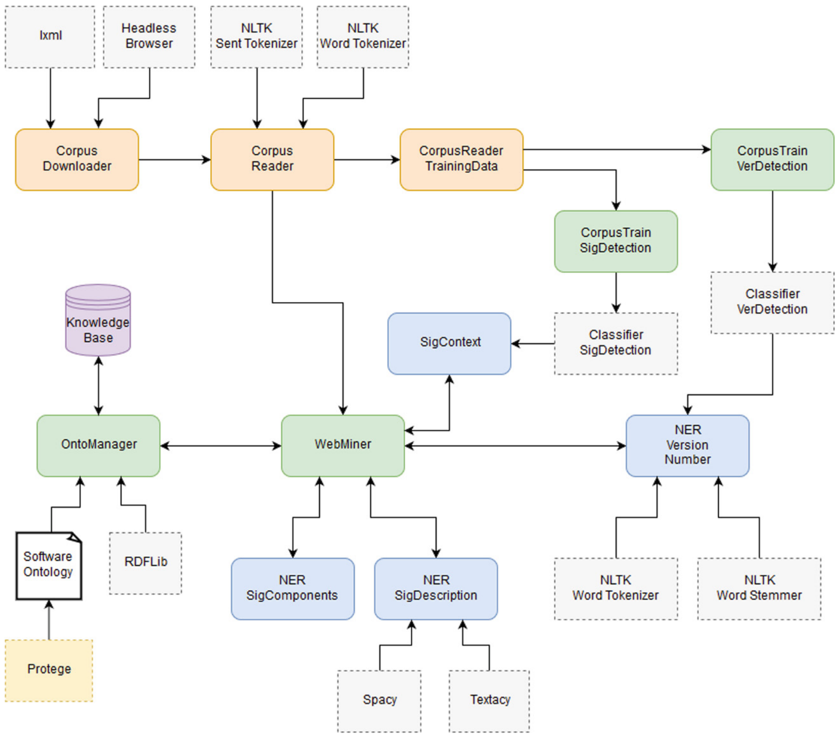


Fig. 2. System architecture. (Source: [33]).

each function is extracted by *SigDescription*, which searches for this information in the descriptive contexts. Finally, all the extracted information is sent to *OntoManager*, which manages the knowledge base. Further, we present the technical details about the implementation of each component.

3.1 CorpusDownloader

This component represents a specialized Web crawler that accesses the online manuals. It uses *lxml* [25], a library that processes XML documents and, implicitly, HTML ones. In the current version, the crawler uses *XPath* expressions to navigate through manuals, which are custom build for each site, pointing it exactly to the pages that contain information about functions. We implemented it like this because we wanted to access only the pages that contain information we are interested in, we did not want to analyze pages containing other data. In future versions, this crawler will be generalized, being able to start from the main page and navigate through the entire manual, selecting the pages that are of interest. The content of each page is then downloaded and saved to a local storage using *Lynx* [11]. This is a text web browser, but we use it as a *headless browser*, without requiring any user input, to save the rendered content, exactly like an user sees it. This approach is an optimization that improves the extraction process. The problem with libraries like *lxml* or *BeatifulSoup*, which are able to download web pages and clean the HTML tags, providing only the content, is generated by the fact that there are differences between how the content is written in HTML and how it is rendered. They leave the content written and positioned exactly how it was inside the tags and there are situations when a sentence (e.g., a function signature) is split into multiple rows in HTML and after removing the tags, it remains written on multiple rows, being way harder to detect it. This depends on how the developers wrote the HTML code. By using a headless browser, the advantages are that our component is independent of the structure of the page, of its stylization, and, furthermore, it is not affected from future layout changes. It gives us the content exactly like a human being is seeing it. The only dependency is towards reaching the desired pages.

3.2 CorpusReader

This component loads the content that is provided by *CorpusDownloader* and tokenizes it. Tokenization represents an important preprocessing step in a standard NLP pipeline. It refers to splitting a text into a set of tokens (e.g., sentences, words, etc.). We split our content into sentences and, afterwards, each sentence is split into words. The order of sentences and words is maintained. The sentence segmentation is done using Punkt sentence segmenter from NLTK platform [28]. It contains an already trained model for English language. This model is built using an unsupervised machine learning algorithm. For word segmentation, Penn Treebank Tokenizer is used (also provided in NLTK), which uses regular expressions to tokenize text. It uses spaces and punctuation signs to separate the words. After that, all punctuation marks are removed, because, in the current version of implementation, they do not bring any valuable information for our extraction process, thus optimizing memory consumption.

3.3 CorpusReader TrainingData

This module loads the training data that is used for generating the classifiers that can identify signatures and version numbers. It uses all the functionalities provided by *CorpusReader* and, additionally, categorizes each instance based on its label, which can be one of the following two: *pos* or *neg*. For the classifier that detects signatures, the instances are stored in a list containing *(sentence, label)* tuples. The training set is manually created with examples taken from PHP and Python manuals and contains 575 instances, 279 positive examples and 296 negative examples. The negative ones were not randomly chosen, we wanted them to contain sentences in which there are parenthesis or different blocks of code where functions are called or defined, because the trained classifier must be able to filter out these cases. For the classifier that detects version numbers, the instances are stored in a list of lists, each sub-list being a sentence in the form *(word, label)* tuples. The training set is manually created with examples taken from PHP, Python, and jQuery manuals, and contains 518 instances, 62 being positive examples and 456 negative. Again, the negative ones were not randomly chosen, they contain various numbers that do not represent a version.

3.4 CorpusTrain SigDetection

This component trains a binary classifier for detecting signatures. The training data from which the feature sets are obtained is provided by *CorpusReader TrainingData*. The assigned labels are *pos* and *neg*. For a signature we use the following features:

– if the character before last is ')' parenthesis;
– if it contains an equal sign before first occurrence of '(' char;
– if it contains special keywords (like *if*, *for*, *foreach*, *while*, etc.) before first '(' char;
– the number of colons before first '(' char;
– if there are more than three words before first '(' char;
– if there are letters before first '(' char;
– if it contains unusual words (like ones that do not contain letters or contain only one character) before first '(' char;
– if the last char is in a predefined list of chars (like !, @, &, *, etc.);
– if after last ')' char there are more than one characters;
– if there is a single pair of top level brackets;
– if there are more than one top level bracket pairs.

We have also tested other features in various combination with the current ones, through a trial-and-error process, but the performance was lower. The tested features are: the last character in the sentence, number of parenthesis groups, if the last character is ')', the character before the last one, the number of words before the first '(' character, the total number of parenthesis.

The training data contains examples taken from PHP and Python manuals. For training, we randomize and split the feature sets, the first 70% being used for training and the rest for testing. The generated feature sets are used to train a Naive Bayes classifier. Although it is one of the simplest models, we have chosen Naive Bayes because it is known to give good results for various use cases and because it does not need a large amount of training instances. The model has an accuracy of approximately 98% on the test set. The generated classifier is used further by *Classifier SigDetection* component, which offers a single service: it receives a sentence and establishes if it represents a signature or not.

3.5 CorpusTrain VerDetection

This module also trains a Naive Bayes binary classifier for detecting version numbers. The feature sets are obtained from the data provided by *CorpusReader TrainingData*. The assigned labels are also *pos* and *neg*. We analyze each word from each sentence, taking also into consideration the word's context. The following features are used for learning:

- if the first character is ‘v’;
- the number of existing dots;
- if most frequent chars are digits;
- the word before;
- if the word before is in a gazetteer list (which contains names of programming languages).

Through a trial-and-error process, we have tested other features in combination with the current ones, but we obtained lower performance. The tested features are: the first character of the word and the 2nd word before.

The training data contains examples from PHP, Python and jQuery manuals. The feature sets are randomized and split, the first 70% being used for training and the rest for testing. The model has an accuracy of approximately 97% on the test set. The generated classifier is used by *Classifier VerDetection* component, which offers a single service: it receives a sentence and a word in that sentence, and establishes if the word represents a version number.

3.6 SigContext

This component receives from *WebMiner* all sentences of the entire document and uses the trained classifier to identify signatures. Afterwards, for each signature, it establishes its descriptive context. We define a descriptive context as the list of sentences that refer to a single function and contain various information about it. The algorithm for identifying the context is intuitive and models the way a person analyzes the manual page: usually all the descriptive details are after the line containing the signature or there are situations when there is a line containing only the function’s name (as is the case with PHP manual). This marks the beginning. The context ends when a line containing another signature is identified or the end of document is reached. *SigContext* builds a model of the document containing indexes that represents the positions of the signatures, the start, and the end of their contexts.

3.7 NER Version Number

This component is capable of identifying the version numbers in which the functions are supported or unsupported. Its sources of information are the list of sentences and the model generated by *SigContext*, received from *WebMiner*. First we check to see if there is a version mentioned outside the descriptive contexts, because there are situations when it is written at the beginning of the document, meaning that the version applies to all the functions. If multiple versions are detected, we established the rule of choosing the first one. It is not the best criterion, but based on a human analysis on different manuals, it proved to offer acceptable results. Afterwards, we look inside each context. Every version detected here will have a higher priority than the one detected outside the contexts. For identifying the version numbers we use the trained classifier.

For each detected version, we further identify its availability status, which can be one of the following: supported, deprecated, removed. We accomplish this by analyzing the context of each number, searching for various keywords that refer to this, like *changed (in)*, *removed (from)*, *added (in)*, *deprecated (since)*, etc. For that, we have built a dictionary containing the stems of these keywords. Then each word in the context of the version is stemmed and compared with the entries in the dictionary in order to detect the state. Stemming is the technique that removes affixes from a word. We use it as an optimization, allowing us not to store all the forms of words in the dictionary. For this procedure, the Porter stemming algorithm provided by NLTK is used.

3.8 NER SigComponents

This module receives from *WebMiner* a sentence representing a function and identifies its components. The following elements are extracted:

- the function's name;
- the function's return type;
- the list of parameters;
- for each parameter, its type and order.

We defined several regular expression and hand-written rules for this extraction.

3.9 NER SigDescription

This component is capable of identifying a function's short description, which offers an explanation of its functionality. It receives the signature and its descriptive context from *WebMiner* and searches for the description by analyzing the grammatical structure of each sentence. The analysis is done using a dependency parser provided by *spaCy* library [15]. This library outputs SVO triples (subject-verb-object), which are further managed by *textacy* library [10]. The source of information that our identification algorithm works with consists of the detected lexical items and their grammatical functions. This algorithm implements some observed patterns commonly used to express descriptions and selects the first sentence that meets its rules. This represents a good choice because the description is usually near the signature. We first look at the subject linked to the root verb. If it contains the function's name, then the sentence is very probable to be the description. If the root verb does not have a subject, we look at its tense. If it is labeled as VB (base form) or VBZ (3rd person singular present), then it is very likely to be the description, this being a very common way used to express it. Also large sentences are split and each part analyzed individually, because of the errors of the dependency parser in such cases, in identifying the root verb, its subject, etc.

Table 3. Signature detection performance. (Source: [33]).

Man page	No. of functions	Detection rate
Node.JS	26	100%
Ruby	59	64.4%
PHP	1	100%
Python	30	100%
Laravel	80	98.75%

3.10 WebMiner

This is the orchestrator of the entire information extraction process. First, it obtains the content of the manual page(s) from *CorpusReader*. The list of sentences is sent to *SigContext*, which provides a model containing indexes of the positions of signatures, the start and the end of their descriptive contexts. Each signature is sent to *NER SigComponents* for obtaining its components. *NER Version Number* receives the contexts and extracts the versions. Each context is also sent to *NER SigDescription* to obtain the signature's short description. Finally, all data is put together and is sent to *OntoManager*.

3.11 OntoManager

This is the component that manages the data in the knowledge base. It receives the extracted information from *WebMiner* and generates instances for the knowledge base, taking care of not creating duplicates. The data is expressed using RDF (Resource Description Framework) model. We manipulate the RDF triples using *RDFLib* [46]. These triples contain information structured according to concepts illustrated by our software ontology. This ontology was created using Protege. We have created this ontology because the existing ones that are specific to our addressed domain, like *SEON Code Ontology* [62], *Core Ontology of Programs and Software (COPS)* [21] and *Core Software Ontology (CSO)*, with its extension, *Core Ontology of Software Components (COSC)* [31], do not contain all the conceptual descriptions that are needed in our case. Thus, we have extended them with some new concepts, attributes and relations that model accurately the extracted information.

4 Experimental Results

The performance of the platform for the current targeted use cases is given by the capability of the classifiers used to detect the signatures and the version numbers. As a validation test, we have pointed the platform to extract data from various pages selected randomly from Node.JS (version 7.7.0) [29], Ruby [48], PHP [35], Python (version 3.6.0) [44], and Laravel [22] online manuals. The obtained results are summarized in Table 3. The second column contains the total number of functions that exist in each page. The last column represents the percentage of the detected signatures. Considering the results, we can say that:

– in case of Node.js, all functions were extracted. We did not have any false positives. All other existing functions (in the code samples and in the menu of the page) were correctly filtered;
– in case of PHP, it correctly identified and extracted the function which is of interest. Unfortunately, it also extracted 5 more functions (false positives), because there are many code examples in the page and it did not succeed in filtering all of them;
– in case of Python, all functions were extracted. We did not have any false positives. Even though other mentions to functions exist in code samples and in the menu of the page, they were successfully filtered;
– in case of Ruby, it only detected 38 functions out of a total of 59. This not very good performance is due to the fact that in the respective page there are functions which do not have parenthesis (e.g., *binding*, *fork*, *chop*), this being a very important feature;
– in case of Laravel, it detected all the functions, except of one, whose name contains a single character (*e()*). We did not have any false positives.

For all detected signatures, the module *NER SigComponents* successfully extracted all of their elements (the name, the return type, the parameters, etc.). Regarding the extraction of version numbers, we obtained the following results:

– for Node.JS: there are 26 functions on the page, 5 of them have specified a single version (representing when it was added) and each of the others have specified 2 versions (when it was added and since when it was deprecated). The platform correctly identified all of them, with their status. We do not have any false positives or negatives;
– for Ruby: there are no mentions to version numbers on the page, thus the system correctly did not identify any;
– for PHP: the page contains a single function with 3 version numbers mentioned. The platform successfully identified all of them;
– for Python: the page contains 30 functions, 4 of them having specified two versions, 1 of them having specified 3 versions, and the rest only 1 version. The system correctly identified all of them, without any false positives or negatives;
– for Laravel: the page does not mention any details about versions inside the context of the functions, thus the system correctly did not identify any. However, there is a version number specified outside the descriptive contexts, which the platform correctly identified.

Regarding the extraction of the short descriptions of functions, the obtained results are summarized in Table 4. The second column contains the number of signatures that were detected, each having a single description. The last column represents the percentage of the identified descriptions. In case of Node.JS, it

Table 4. Description detection performance. (Source: [33]).

Man page	No. of det. functions	Detection rate
Node.JS	26	76.9%
Ruby	38	68.4%
PHP	6	100%
Python	30	93.3%
Laravel	79	96.2%

failed to identify 6 descriptions. For Ruby, it missed 12 descriptions. Regarding PHP, it successfully identified the description of the single function. For the other 5 false positives, it did not detect anything because they are examples of code, thus they do not have descriptions. In case of Python, it did not identify the descriptions of 2 functions. Finally, for Laravel, it failed to extract 3 descriptions.

5 Related Solutions

Our approach has contributions in two directions: automating the compatibility check and extracting entities in the programming domain. Various solutions that address these problems exist, but with major differences. There are tools that were developed with the same purpose as our solution, to verify the compatibility of a new software dependency version, but they are language dependent, are not that general, and require a lot of manual work to create their source of knowledge. For example, the most known and used tools for PHP are PHP CodeSniffer [54] and PHPCompatibility [17]. The former is able to tokenize the source code and to check for violations of code formatting standards, without executing the code. The latter uses this functionality and defines a set of sniffs for verifying version compatibility. A sniff is a code standard for PHP CodeSniffer, created with a different purpose, namely detecting the use of backwards incompatible code. This tool generates a report containing all the identified problems. The advantage over our solution is that it covers more types of changes that occur between versions, not only available/deprecated/obsolete functions. The major disadvantages are that its knowledge base (the sniffs) is created manually, it supports only PHP, and only the changes introduced since version 5.0. Other tools exist for different languages and libraries, like *jquery-migrate* [19] (migrates older jQuery code to jQuery 3.0+), *wp-deprecated-checker* [57], (verifies the use of deprecated functions in Wordpress), *2to3* [63] (automatically translates Python 2 code to Python 3), *PHP7MAR* [39] or *php7cc* [38] (they are PHP 7 migration assistants). Again, the problems with all of them is that they require manual effort to create their knowledge base, have restricted coverage and are built for a specific language or library. More tools exist, but not for every language/library.

Another category of tools is represented by professional-grade IDEs, which offer automatic code inspection functionalities. One such example is PhpStorm [41], which provides a static code analyzer that checks for various code inefficiencies. It can check for: probable bugs, code that never gets executed, possible performance issues, violations of coding guidelines and standards, conformance to specifications, and others. All these inspections are manually created. It does not focus on detecting code compatibility problems, thus support for this kind of functionality is very limited, given by the existence of manually created inspections, specifically for this situations. A third category of tools is represented by "linters" (e.g., *PHPMD* [40], *Pylint* [45]). They provide code analysis functionality, but their focus is not on verifying if deprecated or removed functions are used. They check for different coding standard violations and various errors in how the code is written. In [32] we defined a criteria and presented a comparison of all these tools with our solution.

There is also a lot of research conducted towards automatic verification of backward compatibility problems [1,43,55,61]. These approaches do not provide the same functionality as our solution, they offer a complementary one. They monitor and analyze software repositories, which represent their sources of information. The analysis is done by using different techniques that evaluate the differences between the old source code and the new one (entire code or only the interfaces). The focus is on assessing if a new version of a software component is compatible with its old version, from a functional point of view. They do not mention anything regarding the capability to check for the use of missing functions or to warn about functions that are marked as deprecated.

Our proposal has also a contribution regarding the possibility to identify and extract entities in the programming domain. An overview of different entity types which are being addressed by specialized NERs can be found in [27]. The focus is mainly on extracting proper names (e.g., persons, organizations, locations) and numeric expressions (e.g., time, date, money, percent). Fine-grained subcategories are addressed, like "politician" and "entertainer" for "person" [50], or "city", "state", "country" for "location", or, for specific needs, types like "film", "scientist", "email", "phone number", "research area", "project name", "book title". There is also a lot of research in the bioinformatics fields, where there is the need to extract entities like "protein", "DNA", "RNA", "cell type", "cell line", etc. Another category of NER systems, named open domain NERs [3,14], were developed to extract any type of entity. Although they can be used for any domain, they were tested and fine-tuned for the most well known entity types in the newspaper domains. For these kind of systems an extended named entity hierarchy was proposed [52], containing various categories, but not a single one referring to the programming domain. We did not find any NER system that can extract entities in this domain.

Known commercial solutions that offer text analysis capabilities, like IBM Watson (which integrated AlchemyAPI), OpenCalais, MeaningCloud, recognize general terms, from categories like people, places, dates, not entities in the programming context. We have tested these products [32] on some manual pages chosen randomly from PHP [35], Python [44], and Node.js [29]. In the case of the PHP page, IBM Watson and MeaningCloud could not extract anything, while OpenCalais detected the programming language name and some other terms, but nothing relevant for our purpose. From the Python page, IBM Watson detected the terms “unc” as an organization and “r” as a person, OpenCalais and MeaningCloud detected various IT terms, but still nothing relevant. Same results were obtained in the case of the Node.js page, where neither product was able to extract something relevant.

6 Additional Applicability

Our solution directly targets the persons involved in assuring software administration and software maintenance, for the use cases described previously. The features offered by the platform can also be used in the context of the Semantic Web [33]. The purpose of Semantic Web is to make the content available on the Web understandable not only by humans, but also by computers, without using artificial intelligence. This is mainly done by enriching the Web documents with semantic markup in order to add meaning to the content. Thus, a machine will be able to process knowledge about text. There are many semantic annotation formats that can be used in HTML documents, like Microformats, RDFa, and Microdata. In order to facilitate the annotation process, many systems were developed [6,20,47,51,56]. Considering the annotation process, there are tools that allow users to manually create annotations, and tools that create them semi-automatically or automatically. In latter case, considering the methods used, they can be classified in two categories, pattern-based and machine learning-based (using supervised or unsupervised learning techniques). The tools that are based on supervised learning require training in order to identify our specific types of entities. Also the majority of them require the existence of an ontology that defines the semantics of the domain. To the best of our knowledge, an ontology for the kind of information we deal with does not exist. Also, the existing approaches typically cover real world entities. Our platform provides such an ontology and also the information extraction techniques needed to identify and extract the entities that must be annotated (e.g. the function, return type, parameters, parameters type, etc.). Thus, the capabilities of the platform can enable automatic generation of semantic markup for specific Web documents. A special tool must be developed that is able to match each entity extracted by our platform with the information on a page, adding the appropriate annotations.

Another use case is more advanced and represents a vision [33]. It refers to assisting developers in the process of building new applications, especially for the Internet of Things (IoT). In the context of the IoT, there is the need for a single software application to have support for different devices from different manufacturers. This capability is obtained usually through the implementation of adapters, which are dedicated software modules for each device. Each module uses the specific API (Application Programming Interface) of the vendor to interact with its device. In the case of supporting devices from different manufacturers that have the same functionality (e.g. smart plugs, air quality monitors, dimmer switches, thermostats, smoke detectors, etc.), each API is learned and similar adapters are built, because the functions that interact with the devices are mostly the same, like setting a certain value for a threshold or getting a certain value. The differences are in their name and possibly parameters, but their functionality is the same. Our idea implies the development of a single adapter and the use of a special built tool that is able to analyze the functions used in that adapter and propose the equivalent functions in the APIs of other similar devices for which adapters must be developed. Our knowledge base, that can contain all the functions in each API, is the source of information. Based on that information, we envision the development of a new technology that is able to suggest similar functions in the targeted APIs, by analyzing a function's signature, and most importantly, the meaning of its description and, eventually, each parameter's description. This capability can be achieved with our platform by using NLP resources, like WordNet, a semantically oriented dictionary of English, and techniques for analyzing the structure of a sentence (by using context free grammars, dependency grammars, feature based grammars, etc.) and its meaning (by using first order logic, λ calculus, etc.) [5]. To accomplish semantic analysis is very hard. It represents the most complex phase of natural language processing. There is a lot of ongoing research towards this goal and on the task of computing sentence similarity [4,8,9,13,18,24,26,50]. There are also many tools available capable of comparing the meaning of two sentences, with different success rates [7,12,23,42,49,53]. At the moment, we think that existing solutions can provide acceptable results, but still much further research must be done to accomplish our vision. To the best of our knowledge, there is no available system that is using this kind of techniques and is capable of generating code in this context.

7 Conclusions and Further Work

Keeping software applications and their execution environments up-to-date comes with some challenges. When an update for a software dependency (e.g., library, interpreter) is available, we do not know if the applications will work flawlessly with the new version. Some assumptions can be made, depending on whether it is a minor or major update, but they are not a certainty. This problem is very serious in the case of applications developed using interpreted languages, because the errors appear at runtime, not initially when everything is compiled, as is the case of compiled languages. In this paper we addressed the first kind of applications. In order to support the new version, an assessment must be made regarding the changes that need to be done in an application. This is a very time consuming task. We have proposed a novel approach that helps reduce the needed time, thus improving productivity, by automatically analyzing the source code and assessing the impact that the new release will have on the functionality of the application. Our solution is based on three independent components: a static code analysis tool, a knowledge base containing information about supported functionalities in every version of a targeted library/interpreter, and an extraction platform that manages the data in the knowledge base. In this paper the focus was on the platform's prototype. It does not depend on any programming language and it is based on an automated process that models the human approach for learning an API from a manual. As we have seen, our proposal has applicability in many fields, like system administration, software maintenance, and project management. To the best of our knowledge, similar solutions do not exist. The tools that were developed to address the same problem are language dependent, are limited in coverage, and have their source of information manually created.

Our approach is based on many functionalities, having varying degrees of complexity, some even requiring further research. Improvements can be made to any of them. Regarding the platform, updating the headless browser is considered, in order to be aligned with the latest HTML and CSS specifications, for presenting the content adequately. Improvements to the performance of the classifiers will be made, by refining the training and validation data sets. Other features will also be implemented and various combinations of them will be tested. Besides Naive Bayes, we plan to use other models to try to improve the performance, with a special focus on artificial neural networks. Our efforts will also be directed towards the development of the static code analysis tool.

References

1. Ahmed, H., Elgamal, A., Elshishiny, H., Ibrahim, M.: Verification of backward compatibility of software components. US Patent 9,424,025 (2016)
2. Alboaie, S., Alboaie, L., Panu, A.: Levels of privacy for ehealth systems in the cloud era. In: Proceedings of the 24th International Conference on Information Systems Development, pp. 243–252 (2015)
3. Alfonseca, E., Manandhar, S.: An unsupervised method for general named entity recognition and automated concept discovery. In: Proceedings of the 1st International Conference on General WordNet (2002)
4. Atoum, I., Otoom, A., Kulathuramaiyer, N.: A comprehensive comparative study of word and sentence similarity measures. Int. J. Comput. Appl. **135**(1), 10–17 (2016)
5. Bird, S., Klein, E., Loper, E.: Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media Inc., Newton (2017). http://www.nltk.org/book/
6. Charton, E., Gagnon, M., Ozell, B.: Automatic semantic web annotation of named entities. In: Butz, C., Lingras, P. (eds.) AI 2011. LNCS (LNAI), vol. 6657, pp. 74–85. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21043-3_10
7. Cortical.io: Similarity Explorer (2017). http://www.cortical.io/similarity-explorer.html
8. Crockett, K., McLean, D., O'Shea, J.D., Bandar, Z.A., Li, Y.: Sentence similarity based on semantic nets and corpus statistics. IEEE Trans. Knowl. Data Eng. **18**(8), 1138–1150 (2006)
9. Dao, T.N., Simpson, T.: Measuring similarity between sentences. WordNet.Net, Technical report (2005)
10. DeWilde, B.: Textacy NLP library (2017). http://textacy.readthedocs.io/en/latest/
11. Dickey, T.E.: Lynx Text Web Browser (2017). http://lynx.browser.org/

12. DKPro: DKPro Similarity Framework (2017). https://dkpro.github.io/dkpro-similarity/
13. Erk, K.: Vector space models of word meaning and phrase meaning: a survey. Lang. Linguist. Compass **6**(10), 635–653 (2012)
14. Evans, R.: A framework for named entity recognition in the open domain. In: Recent Advances in Natural Language Processing III: Selected Papers from RANLP, vol. 260, pp. 267–274, 110 (2003)
15. ExplosionAI: Spacy NLP library (2017). https://spacy.io/
16. Ferretti, E.: Privacy by design and privacy by default (2015). http://europrivacy.info/2015/06/09/privacydesign-privacy-default/
17. Godden, W: PHPCompatibility (2017). https://github.com/wimg/PHPCompatibility
18. He, H., Gimpel, K., Lin, J.: Multi-perspective sentence similarity modeling with convolutional neural networks. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1576–1586 (2015)
19. jquery-migrate (2017). https://github.com/jquery/jquery-migrate
20. Laclavik, M., Seleng, M., Ciglan, M., Hluch, L.: Ontea: platform for pattern based automated semantic annotation. Comput. Inform. **28**(4), 555–579 (2012)
21. Lando, P., Lapujade, A., Kassel, G., Furst, F.: Towards a general ontology of computer programs. In: Proceedings of the International Conference on Software and Data Technologies, pp. 163–170 (2007)
22. Laravel: Manual page (2017). https://laravel.com/docs/5.4/helpers
23. Linguatools: DISCO (2017). http://www.linguatools.de/disco/
24. Liu, H., Wang, P.: Assessing sentence similarity using wordnet based word similarity. JSW **8**(6), 1451–1458 (2013)
25. lxml: lxml XML toolkit (2017). http://lxml.de/
26. Miura, N., Takagi, T.: WSL: sentence similarity using semantic distance between words. In: Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), Denver, Colorado, pp. 128–131. Association for Computational Linguistics (2015)
27. Nadeau, D., Sekine, S.: A survey of named entity recognition and classification. Lingvisticae Investigationes **30**(1), 3–26 (2007)
28. NLTK: Natural Language Toolkit (2017). http://www.nltk.org/
29. Node.js: Manual page (2017). https://nodejs.org/api/util.html
30. Noy, N.F., McGuinness, D.L., et al.: Ontology development 101: a guide to creating your first ontology (2001)
31. Oberle, D., Grimm, S., Staab, S.: An ontology for software. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies. IHIS, pp. 383–402. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-92673-3_17
32. Panu, A.: Automation technology for software maintenance and system administration. Ph.D. thesis. Alexandru Ioan Cuza University of Iasi (2017)
33. Panu, A.: A novel method for improving productivity in software administration and maintenance. In: Proceedings of the 12th International Conference on Software Technologies (ICSOFT 2017), Spain, 24–26 July 2017, pp. 220–229 (2017)
34. Pearson, S.: Taking account of privacy when designing cloud computing services. In: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, pp. 44–52. IEEE Computer Society (2009)
35. PHP: Manual page (2017). http://php.net/manual/en/function.chmod.php
36. PHP: Past Releases (2017). https://secure.php.net/releases/
37. PHP: Supported Versions (2017). http://php.net/supported-versions.php

38. php7cc (2017). https://github.com/sstalle/php7cc
39. PHP7MAR (2017). https://github.com/Alexia/php7mar
40. PHPMD (2017). https://phpmd.org/
41. PhpStorm (2017). https://www.jetbrains.com/phpstorm/
42. Pilehvar, M.T., Jurgens, D., Navigli, R.: Align, disambiguate and walk: a unified approach for measuring semantic similarity. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, pp. 1341–1351. ACL (2013)
43. Ponomarenko, A., Rubanov, V.: Backward compatibility of software interfaces: steps towards automatic verification. Program. Comput. Softw. **38**(5), 257–267 (2012)
44. Python: Manual page (2017). https://docs.python.org/3/library/os.path.html
45. Pylint (2017). https://www.pylint.org/
46. RDFLib: RDFLib RDF library (2017). https://rdflib.readthedocs.io/en/stable/
47. Reeve, L., Han, H.: Survey of semantic annotation platforms. In: Proceedings of the 2005 ACM Symposium on Applied Computing, SAC 2005, pp. 1634–1638. ACM, New York (2005)
48. Ruby: Manual page (2017). https://ruby-doc.org/docs/ruby-doc-bundle/Manual/man-1.4/function.html
49. RxNLP: Text Similarity API (2017). http://www.rxnlp.com/api-reference/text-similarity-api-reference/
50. Sanborn, A., Skryzalin, J.: Deep learning for semantic similarity. In: CS224d: Deep Learning for Natural Language Processing, Stanford, CA, USA. Stanford University (2015)
51. Sanchez, D., Isern, D., Millan, M.: Content annotation for the semantic web: an automatic web based approach. Knowl. Inf. Syst. **27**(3), 393–418 (2011)
52. Sekine, S., Nobata, C.: Deffinition, dictionaries and tagger for extended named entity hierarchy. In: Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, 26–28 May 2004, Lisbon, Portugal (2004)
53. SEMILAR: A Semantic Similarity Toolkit (2017). http://deeptutor2.memphis.edu/Semilar-Web/public/contact.html
54. Sherwood, G.: PHP CodeSniffer (2017). http://pear.php.net/package/PHPCodeSniffer/
55. Tsantilis, E.: Method and system to monitor software interface updates and assess backward compatibility. US Patent 7,600,219 (2009)
56. Uren, V., Cimiano, P., Iria, J., Handschuh, S., Vargas-Vera, M., Motta, E., Ciravegna, F.: Semantic annotation for knowledge management: requirements and a survey of the state of the art. Web Semant. **4**(1), 14–28 (2006)
57. wp-deprecated-checker (2017). https://gist.github.com/jbuchbinder/7419000
58. W3Techs - World Wide Web Technology Surveys (2017). https://w3techs.com/
59. W3Techs: Usage statistics and market share of PHP for websites (2017). https://w3techs.com/technologies/details/pl-php/all/all
60. W3Techs: Usage statistics and market share of PHP version 5 for websites (2017). https://w3techs.com/technologies/details/pl-php/5/all
61. Welsch, Y., Poetzsch-Heffter, A.: Verifying backwards compatibility of object-oriented libraries using Boogie. In: Proceedings of the 14th Workshop on Formal Techniques for Java-Like Programs, FTfJP 2012, pp. 35–41. ACM (2012)
62. Wursch, M., Ghezzi, G., Hert, M., Reif, G., Gall, H.C.: SEON: a pyramid of ontologies for software evolution and its applications. Computing **94**(11), 857–885 (2012)
63. 2to3 (2017). https://docs.python.org/3/library/2to3.html

From Specification to Implementation of an Automotive Transport System

Oussama Khlifi[1,2,4(✉)], Christian Siegwart[2], Olfa Mosbahi[3], Mohamed Khalgui[3,5], and Georg Frey[1,2]

[1] Chair of Automation, Saarland University, Saarbrücken, Germany
{oussama.khlifi,georg.frey}@aut.uni-saarland.de
[2] ZeMA – Zentrum fur Mechatronik und Automatisierungstechnik gemeinnützige GmbH, Saarbrücken, Germany
c.siegwart@zema.de
[3] LISI Laboratory, INSAT, University of Carthage, Tunis, Tunisia
olfamosbahi@gmail.com, khalgui.mohamed@gmail.com
[4] Polytechnic School of Tunisia, University of Carthage, Tunis, Tunisia
[5] School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China

Abstract. Reconfiguration is often a major undertaking for systems because it can violate memory usage, the required energy and the concerned real-time constraints. The languages in which adaptive probabilistic systems are specified should be clear and intuitive, and thus accessible to generation, inspection and modification by humans. This paper introduces a new specification approach for adaptive probabilistic discrete event systems running under resources constraints. The semantics of the formalism GR-TNCES are presented to optimize the specification approach and applied to specify the requirements of an automotive transport system to prove its relevance. Then, we model, simulate and implement the proposed case study.

Keywords: Requirement specification · Adaptive systems · Statecharts Modeling

1 Introduction

Probabilistic reconfigurable systems have ability to change their behaviors during run-time process to cope with unpredictable significant changes at runtime such as component failures. A system is nondeterministic if the set of enabled transitions is not unique, i.e., some machines can have more than one enabled transition at the same time. A reconfiguration/adaptation scenario is any automatic run-time operation that adds/removes hardware/software components in the system. It can also modify the connections between them, and possibly change the states in response to errors or satisfy unpredictable user requirements. Thus, it is often a major issue for some critical systems and other intelligent systems. Examples of reconfigurable systems include different space systems, control plants and interactive software of varying nature [1]. A reactive system is not adequately described by a simple relationship that specifies

E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 49–68, 2018.
https://doi.org/10.1007/978-3-319-93641-3_3

outputs as a function of inputs, but, rather, requires relating outputs to inputs through their allowed combinations in time [2]. A variety of approaches and methods ranging from model checking to static analysis, simulation and theorem proving are used to ensure and prove the correctness of a system specification. A state-based description of a system is assessed with respect to a properties expressed in an appropriate specification language, e.g., temporal logic [3].

System specification is an important part of system design, thus the language used to specify the system should be expressive enough to allow the designer to encode the system constraints and functionalities. This is important especially for embedded systems which are often used in critical real-time applications, i.e., if it is not possible to specify these hard constraints, then, this specification language is useless in our tasks. Thus, specification should be clear and intuitive, i.e., precise and conscientious to ensure the analysis and simulation by computers [4]. Such a method should make it possible to move easily with sufficient semantic from the initial stages of requirements and specification to prototype, design, and to form the basis for modifications and maintenance [5]. The included behavioral and control aspects should be based on large extent of visual formalisms and admit a formal semantics that provides a precise and unambiguous meaning [6]. For probabilistic reactive systems, the specification method should deal with probabilistic reconfiguration under resources constraints, i.e., the resources availability is not guaranteed after an adaptation process. Statecharts [7] and temporal logic are currently used to specify various systems. Nevertheless, statecharts are not able to specify reconfigurable probabilistic behavior and time constraints. Moreover, temporal logic could not easily deal with unpredictable reconfiguration scenarios during run-time process. It is not a trivial activity to specify reconfigurable processes under limited energy and memory resources. These kinds of systems are considered critical because such a system can violate its resources after any adaptation scenarios [8].

The language used to specify the system must be expressive enough to allow the designer to encode the system constraints and functionalities. In particular, a system specification affects the safety and correctness of probabilistic adaptive systems, i.e., it is used for the requirements specification and the formal verification. Typically, such descriptions involve complex sequences of events, actions, conditions and information flow, often with explicit timing, energy and memory constraints that combine to form the overall behavior of a system [1]. This paper focus on the specification of systems, those are able to undergo structural changes, i.e., it introduces a specification approach based on GR-TNCES formalism “Generalized Reconfigurable Timed Net Condition Event Systems” which enables us to cover the limits of statecharts and temporal logic. We describe also how to encode system specification and its requirements with an optimized and expressive approach. There are many systems which are operating under energy and memory constraints [9], thus, the designer has to optimize the resources consumption for an energy efficiency perspective. Moreover, the paper tries to present a complete approach ranging from specification, modeling, and simulation to the implementation of an automotive transport system. The authors specify the proposed system with the aim to save energy in the skid conveyor and present the system model using the environment ZIZO which is used for system modeling and simulation respecting the GR-TNCES formalism [10]. The implementation of the proposed model is also described in this paper.

The remainder of this paper is organized as follows. The next Section describes the preliminaries on top of system analysis and specification approach. Section 3 introduces the new semantics of the proposed specification. The case study and the system's model are introduced in Sect. 4. Then, the implementation part is given in Sect. 5. A discussion is provided in Sect. 6 and finally, Sect. 7 concludes the paper.

2 Background

In this section, we introduce the syntax and semantics of R-TNCES, GR-TNCES and statecharts. We present also an approach used for system analysis.

2.1 System Analysis

Adaptive probabilistic systems under development should be specified and analyzed [11] from three closely related points: functional, behavioral and structural [4]. In the structural view, the hierarchical decomposition of the system under development into its components is provided. We present also the information that flows between them, e.g., data and control signals. Nevertheless, we do not specify when it will flow, how often and in response to what. The functional view can identify a specified hierarchy of active and exchanged signals between them. However, we do not specify the dynamics, i.e., when the activities will start, whether or not they terminate, and whether they can be carried out in parallel. In the functional view, we only specify that the data can flow and not when it will terminate [4]. In other words, the functional view presents the decomposition into activities and the possible flow of information, but not how those activities and their associated inputs and outputs are controlled during the continued behavior. It is the behavioral view [4] that is responsible for specifying the control tasks, i.e., this is achieved by allowing a control activity to be presented on each level. These controllers are responsible for specifying when, how and why things happen as the system reacts over time.

2.2 Related Work

There have been a set of approaches for formal specification of different systems, e.g., the state/event approach, in the form of finite-state machines or state transition diagrams, has been suggested numerous times for system specification. It proposes state machines for the user interface of interactive software, data-processing systems, hardware system description, communication protocols and computer-aided instruction [12]. There is also augmented transition networks used for hierarchical state/event descriptions by authorizing a transition in one machine to be labelled using another machine's name [13]. Various methodologies were proposed for the specification of complex systems, such as SADT [14] that focus on the system's functional and structural aspects, nevertheless, it does not provide any dynamic semantics related to their behavioral characteristics. Zhang et al. [3] propose a new extension of Petri nets called R-TNCES for reconfigurable systems without considering probabilistic behaviors. There were also different approaches based on formal methods: Zedan et al. [15] present an object based formal method for the development of real-time systems which is called ATOM, i.e., it is based on the refinement calculus and the formal specification that contains a description of the system behavior. An executable specification model [16] was proposed for an abstract transactional memory (lock-free technique) that offers a parallel programming model for future chip multiprocessor systems.

2.3 Statecharts

The statecharts language is presented for specifying complex reactive systems [7]. RSML is another language based on statecharts with slightly different syntax and semantics [5]. They both extend state-machine diagrams with parallelism, superstates, and broadcast communications. The STATEMATE toolset implements a particular semantics of statecharts. It presents a system model which consists of a finite number of parallel local state machines with a finite set of events and inputs interacting with a nondeterministic environment. Figure 1 [7] presents a simple example with two parallel state machines A and B which are synchronized using events. Arrows without sources present the initial local states and the other arrows indicate local transitions, which are identified with the form *trig*[*cond*]/*acts*, i.e., the trig is a trigger event, *cond* is an optional guarding condition, and acts is a (possibly empty) list of action events. The guarding condition is simply a predicate on local states of other state machines and/or inputs to the system. The general idea is that if the trigger event occurs and the guarding condition is either absent or evaluated to true, then the transition is enabled. Initially, some external events, along with some inputs from the environment, arrive, marking the beginning of a step. The system leaves the source local states, enters the destination local states, and generates the action events (if any). The events are used to enable some transitions as described above.

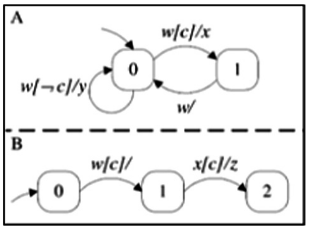


Fig. 1. Statechart example.

2.4 R-TNCES

An R-TNCES, is presented in [3], as a structure *RTN* = (*B*, *R*), where *R* is the control module based on a set of reconfiguration functions $R = \{r_1, \ldots, r_n\}$ and *B* is the behavior module that is a union of multi TNCESs, represented as:

$B = (P, T, F, W, CN, EN, DC, V, Z)$ where: (i) P (*respectively*, T) is a non-empty finite set of places (*respectively*, transitions), (ii) $F \subseteq (P \times T) \cup (T \times P)$ is a subset of flow arcs, (iii) $W : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ maps a flow arc to a weight, $W(x, y) > 0$ if $(x, y) \in F$, and $W(x, y) = 0$ otherwise, where $x, y \in P \cup T$, (iv) $CN \subseteq (P \times T)$ (*respectively*, $EN \subseteq (T \times T)$) is a subset of condition signals (*respectively*, event signals), (v) $DC : F \cap (P \times T) \rightarrow \{[l_1, h_1], \ldots, [l|F \cap (P \times T)|, h|F \cap (P \times T)|]\}$ is a subset of time constraints on output arcs, where $i \in [1, |F \cap (P \times T)|], li, hi \in N$, and $li < hi$, (vi) $V : T \rightarrow \{\vee, \wedge\}$ maps an event-processing mode (*AND* or *OR*) for every transition, (vii) $Z = (M_0, D_0)$, where $M_0 : P \rightarrow \{0, 1\}$ is the initial marking and $D_0 : P \rightarrow \{0\}$ is the initial clock position.

2.5 GR-TNCES

The formalism GR-TNCES is introduced to model and control APDECS running under memory and energy constraints [1]. A GR-TNCES is a network of R-TNCES. It is a structure $G = \sum R$ - TNCES where R - TNCES = (B, R). R is the control module based on a set of reconfiguration functions $\{r1, \ldots, rn\}$ running under memory and energy controllers, and B is the behavior module which is a union of multi TNCES [3], represented as follows: $B = (P, T, F, QW, CN, EN, DC, V, Z_0)$ where:

(i) P (respectively, T) is a non-empty finite set of places (*respectively*, transitions);
(ii) F is a set of flow arcs with $F \subseteq (P \times T) \cup (T \times P)$;
(iii) $QW = (Q, W)$ where $Q : F \rightarrow [0,\ 1]$ is a real number that represents the probability on the arcs and $W : (P \times T) \cup (T \times P) \rightarrow \{0,\ 1\}$ maps a flow arc to a weight. Specifically, $W(x,\ y) > 0$ if $(x,\ y) \in F$, and $W(x,\ y) = 0$ otherwise, where $x,\ y \in P \cup T$;
(iv) CN (respectively, EN) is a set of condition (*respectively*, event) signals with $CN \subseteq (P \times T)$ (*respectively*, $EN \subseteq (T \times T)$);
(v) $DC : F \subseteq (P \times T) \rightarrow [l,\ h]$ is a superset of time constraints on output arcs;
(vi) $V : T \rightarrow \{\vee, \wedge\}$ maps an event-processing mode (*AND* or *OR*) to each transition;
(vii) $Z_0 = (T_0,\ D_0)$ where $T_0 : P \rightarrow \{0,\ 1\}$ is the initial marking and $D_0 : P \rightarrow \{0\}$ is the initial clock position.

Each reconfiguration r is controlled by the controller module R. It is a structure R consisting of a set of reconfiguration functions $\{r1, \ldots, rn\}$. A reconfiguration function r is a structure $r = (Cond, Q, E_0, M_0, S, X)$, where:

(i) $Cond : CN \rightarrow \{\text{true}, \text{false}\}$: The precondition $Cond$ of r could be evaluated to either true or false and could be modeled by external condition signals;
(ii) $Q : F \rightarrow [0..1]$: Represents the probability to reach each TNCES branch. It could be a functional (internal to the TNCES) or a reconfiguration probability. It is used to describe the nondeterministic behavior of the system;
(iii) $E_0 : P \rightarrow [0..\text{max}]$: The energy requirements of the chosen TNCES branch;
(iv) $M_0 : P \rightarrow [0..\text{max}]$: The memory requirements of the chosen TNCES branch;

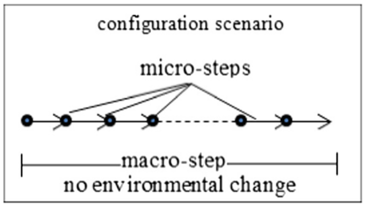


Fig. 2. Macro-step, micro-step.

(v) $S : TN(\bullet r) \rightarrow TN(r\bullet)$: Is the structure modification instruction of the reconfiguration scenario. It contains the reconfiguration structure process, i.e., the information about the current state and the destination;

(vi) X: last state $(\bullet r) \rightarrow$ initial state$(r\bullet)$: Is the state processing function, where last state $(\bullet r)$ (*respectively*, initial state $(r\bullet)$) denotes the last (*respectively*, initial) state of •*r* (*respectively*, *r*•) before (*respectively,* after) the application of *r.*

Let $TN = P \times T \times F \times QW \times CN \times EN \times DC \times V$ be the Cartesian product of all feasible net structures that can be performed by a system. Let •*r* (*respectively*, *r*•) be the source (*respectively*, target) R-TNCES before (*respectively*, after) the reconfiguration function *r* is applied, where $TN(\bullet r)$, $TN(r\bullet) \in TN$. A state machine specified by an R-TNCES called *Structure_changer*, is introduced to describe the control module. In this state machine, each place corresponds to a specific TNCES that refers to a configuration scenario. This place can be introduced as a macro-step which is composed of a set of micro-steps as shown in Fig. 2 [8].

Initially, some external events along some inputs from the environment arrive, marking the beginning of a macro-step. Then, the system leaves the source states, enters the destination local states, and generates the action events (if any). Unless they are regenerated by other transitions, the events disappear after one micro-step and the macro-step is finished if there is no enabled transition. Each transition of the S*tructure_changer* corresponds to a reconfiguration function. A place *sp* gets a token implies that the TNCES to which *sp* corresponds is selected. If a transition $st(\forall st \in sp\bullet)$ fires, then it removes the token from *sp* and brings it into a place *sp*' with s*p'* ∈ *st*•. Firing *st* corresponds to the application of a reconfiguration function. The *Structure_changer* is formalized as follows: $Structure\_changer = (P, T, F, Q, E', M')$ where $\forall t \in T, |\bullet t| = |t\bullet| = 1$, and only one TNCES is performed at any time. Each place of this structure contains the whole information about the corresponding TNCES, i.e., its energy and memory requirements. Let *ß* be a TNCES and Cost TNCES be the needed resources by this TNCES. The states of a GR-TNCES are defined as follows; a state of *G* is a pair (*TN* (*ß*), *State*(*ß*)), where *TN*(*ß*) denotes the net structure of *G* and *State*(*ß*) denotes a state of *G*. The evolution of a GR-TNCES depends on which events, energy and memory constraints take place. A reconfiguration function $r = (Cond, Q, E', M', S, X)$ is enabled at state (*TN*(*ß*), *State*(*ß*)) if the following conditions are met:

(i) *TN*(*ß*) = *TN*(•*r*), i.e., *TN*(*ß*) is equal to the net structure of •*r* and the firing time constraints are valid,

(ii) *Cond* = true: The reconfiguration's precondition is fulfilled,
(iii) The energy reserves E' are enough: i.e., $E' > \text{Cost}\,\text{TNCES}\left(E'_0\right)$,
(iv) The memory reserves M' are enough: i.e., $M' > \text{Cost}\,\text{TNCES}(M_0)$.

The reconfiguration function is a tuple based on the required energy and memory resources compared with the current resources storage as well as the events and conditions signals. To select the most probabilistic reconfiguration scenario R_{Max}, the controller chooses the maximal probabilistic transition to be fired in the next step. Let (i) '$\cap e \in EN\, e$' and '$\cap c \in CN\, c$' be *respectively* the set of all possible *Event-In* and *Condition-In* of the desired transition, (ii) E' and M' are *respectively* the energy and memory reserves, (iii) '$\text{Cost}\,\text{TNCES}_{\text{Max}}\left(E'_0\right)$' and '$\text{Cost}\,\text{TNCES}_{\text{Max}}\left(M'_0\right)$' are *respectively* the energy and memory required by the most probabilistic reconfigurable scenario. The reconfiguration is applied by respecting this formula:

$$R_{Max} \equiv \left(E' > \text{Cost}\,\text{TNCES}_{Max}\left(E'_0\right)\right) \wedge \left(M' > \text{Cost}\,\text{TNCES}Max\left(M'_0\right)\right) \wedge \cap e \in EN\, e \wedge \cap c \in CN\, c \tag{1}$$

Indeed, the highest probabilistic scenario has to guarantee that: (i) the resource constraints related to the energy and memory resources, and (ii) the events and conditions should also occur at the firing time.

3 Specification Approach

To analyze GR-TNCES using state-exploration techniques, we focus separately on the behavior and the control module. We consider the control module as a transition system (C, Rec, In) where C is a set of macro-steps, $Rec \subseteq C \times C$ a transition relation or reconfiguration function. It maps the reconfiguration scenario to the respected constrains (energy, memory, probability). *In* represents the initial standard system configuration, i.e., the start point is a static state. The reconfiguration function is a tuple of the current configuration (macro-step), the corresponding events and conditions, the desired probability, and the needed energy and memory resources compared to the current storage. To execute the highest probabilistic reconfiguration scenario, the controller has to choose the maximum probabilistic transition for the next step respecting this formula:

$$Rec_{Max} \equiv (E' > Cost\,\text{TNCES}_{Max}(E_0)) \wedge (M' > Cost\,\text{TNCES}_{Max}(M_0)) \wedge \cap_{\text{e}\in\text{EN}} e \wedge \cap_{\text{c}\in\text{CN}} c \tag{2}$$

which describes how macro-steps are selected [8], i.e., the highest probabilistic scenario has to guarantee the resource constraints related to energy and memory reserves. Moreover, the events and conditions should also occur, otherwise they are considered to be true. For the low probabilistic reconfiguration, the transition relation will be introduced as follows:

$$Rec_{Min} \equiv (E' > Cost\,\text{TNCES}_{Min}(E_0)) \wedge (M' > Cost\,\text{TNCES}_{Min}(M_0)) \wedge \cap_{e \in EN} e \wedge \cap_{c \in CN} c \tag{3}$$

which describes how the macro-steps are selected [8], i.e., the lowest probabilistic scenario has also to satisfy the resource constraints related to the energy and memory reserves. The events and conditions should also occur, i.e., if there is no event, then the input is considered to be true.

Once the macro-step is selected, the system executes the micro-steps of the selected configuration. The behavior module is considered as a transition system (*P, R, I*) where *P* is a set of global states, $R \subseteq P \times P$ a transition relation. It is a labeling function that maps each transition to the holding properties in the corresponding transition, and $I \subseteq P$ a set of initial states. A transition in *R* is a tuple of the current local state (system source state), the events and conditions occurring, the probabilistic value of the environment inputs and the time period in which the transition could be fired. A path is sequence of states that belongs to *P*, i.e., a state is reachable only if it appears on such trace path execution. We symbolically encode the global state space *P* of a GR-TNCES system using a set of variables *Y* as follows: For each system state *m*, we consider a state variable from the local states of *m*. The set of initial states *I* is represented as:

$$I \equiv \cap_{m \in P} m \equiv m_0 \wedge \cap_{e \in Ei} \neg e \wedge \cap_{c \in CNi} \neg c \wedge (T_0 = \{1\}) \wedge (D_0 = \{0\}) \tag{4}$$

where m_0 is the initial local state, *Ei* and *CNi* are respectively the set of internal events and internal guarding condition [8]. Initially, the system is in its initial local state, all internal events and guarding conditions do not occur, the state is marked and the clock position is null. The most important thing is the encoding of the nondeterministic transition relation *R*. We focus on the encoding of the micro-step transition, i.e., for each state variable $var \in Y$, we present a variable var' that has the same range as *var* and intuitively represents its next-state value. Let Y_0 be the set of all these primed variables. We aim to define an expression over $Y \cup Y_0$ to specify *R*, then for each local transition *t*, let *src*(*t*), *dst*(*t*), *evt*(*t*), *cond*(*t*), *time*(*t*), *mode*(*t*), and *prob*(*t*), be respectively the source local state, destination local state, trigger event, guarding condition, and the firing time interval, the firing mode{*AND, OR*}, and the firing probability. The expression *evt*(*t*) and *cond*(*t*) could be true if the transition *t* does not have a guarding condition and event inputs. Let *curr*(*t*) be the current local state of the system and $enb_{prob}(t)$ to be represented as:

$$enb_{prob}(t) \equiv curr(t) \wedge evt(t) \wedge cond(t) \wedge time(t) \tag{5}$$

It is enabled once the trigger events and guarding conditions occurs simultaneously at the desired running time if the firing mode is '*AND*' [8]. We could deal with other firing mode as described here:

$$enb_{prob}(t) \equiv curr(t) \wedge time(t) \wedge (evt(t) \vee cond(t)) \tag{6}$$

It presents how the transition could be enabled if the firing mode is ‘*OR*’, i.e., it is considered to be true if one trigger event or guarding condition occurs at the required running time period of the selected transition [8]. Once the system executes a configuration scenario, we aim to describe how the system deals with the micro-steps. For each state *m* of the system, $micro_m$ describes the progress of the system at run-time process:

$$micro_m \equiv (\cap_{t/curr(t)=m}(enb_{prob}(t) \rightarrow curr'(t) = dst(t))) \vee (\cap_{t/curr(t)=m}(\neg enb(t) \rightarrow curr'(t) = curr(t))) \tag{7}$$

Indeed, the first conjunct guides the system states from the enabled transition to the destination state, while the second conjunct blocks the system on the same position if none of the transitions are enabled [8]. The fired transition can generate various events, moreover, the generation of events *evt*(*t*) and conditions *cond*(*t*) are introduced respectively as follows:

$$micro_e \equiv \ (\cup_{t/e \in Evt(t)} enb_{prob}(t)) \leftrightarrow e' \tag{8}$$

The event is delivered by the union of the enabled transition that can send events to activate different states of the system [8]. Similarly, the micro-step generates guarding condition and it is represented as:

$$micro_c \equiv \ (\cup_{m/c \in Cnd(t)} micro_m(t)) \leftrightarrow c' \tag{9}$$

It is generated by a union of states to control the execution of various tasks of the system [8]. Then, we introduce *macro* to encode the macro-step which is a conjunction of micro states, micro events and micro conditions as follows:

$$Macro \equiv \cap_{e \in CN} micro_c \wedge \cap_{c \in EN} micro_c \wedge \cap_{m \in P} micro_m \tag{10}$$

The presented specification approach makes possible to deal simultaneously with unpredictable reconfiguration scenario [8], time constraints, and limited energy and memory resources. It is useful to specify the system requirements in an optimized method.

4 Test Case: Skid Conveyer

Skid conveyors are widely used to move materials over a fixed path in the automotive industry. Transporting a body in the paint shop or a chassis from one workstation to another in the final assemblies are typical examples. For this purposes, we define an extended skid conveyor system showed in Fig. 3, which will be one part of the automated commissioning line in the “Zentrum für Mechatronik und Automatisierungstechnik” (*ZeMA*) in Saarbrücken, Germany. The following section describes the functional requirements of the system.

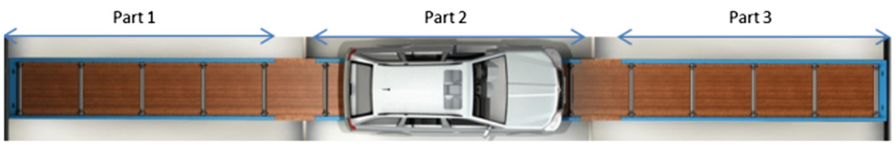


Fig. 3. CAD model.

4.1 Functional Requirements

The transport system should consist of three conveyor parts [17]. Currently, there is an old system where all the motors are switched together and manually from one mode to another. We aim to introduce new functional modes which offer the user to localize the chassis on every part, i.e., in each conveyor part, the chassis should wait for 7 s to establish other tasks by various robots. To minimize the consumed energy of the system during the movement of the chassis, each unused actor should be switched off or to a standby mode. For example once the chassis is in the third conveyor part, the motor of the second one should be switched off. The activation/deactivation of the motors is controlled based on the car position. Moreover, the worker should control the system with a control panel showed in Fig. 4 with all the possible uses cases [8].

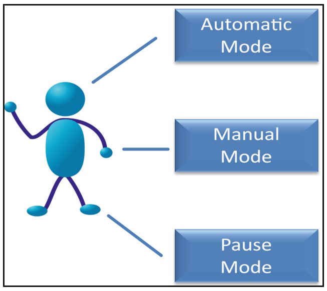


Fig. 4. Worker use cases.

4.1.1 Control

Using this approach, it is possible to choose one operation mode for the system. The requirements for these operation modes are explained in the following part. The system is reconfigurable and we consider three possible reconfigurations:

- **Automatic Mode.** The worker activates and stops this mode using the panel. The speed of the skid should be as well controlled. Then, all other sensors and actors operate automatically, i.e., (i) the chassis position has to be clear, (ii) then, the chassis moves from one workstation to another without user interaction. Since the position of the chassis is logged, all unused actors can be switched *Off*. As soon as the chassis is at the third position it should move backwards to the start position and start again.

- **Manual Mode.** In this mode, the worker should manually control the system's functionalities, i.e., to start and stop all the conveyor parts individually and together and controls the speed of the chassis.
- **Pause Mode.** In order to save energy the worker can activate and deactivate this mode with the panel. If the mode is activated all sensors and actors are switched *Off* or change to a *standby* mode.

4.1.2 Additional Information

If the worker uses this option, he can visualize all the relevant sensor and actor data. For example whether the motor is *ON* or *Off* and its speed.

- **Settings.** This mode should help the worker to use the panel. It should be possible to increase and reduce the contrast or to calibrate the screen.
- **Diagnosis.** If an error occurs, the worker can choose this mode and all sensor and actor errors are displayed here.

4.2 System Encoding

The skid conveyor is supervised by a centralized controller, i.e., it enables to control and switch the system from one configuration to a second one. To simplify the use case specification, we consider that the system is not probabilistic and that the switching mode is chosen by the user to be denoted by $RTN_{skid} = \{B_{skid}, R_{skid}\}$. Let E_{skid} and M_{skid} be *respectively* the energy and memory reserves of the skid. We use the proposed specification approach to specify the system, and then each mode is represented by a macro-step. We identify three macro-steps for the different modes: *Rec1* = *Macro1*: Automatic mode, *Rec2* = *Macro2*: Manual mode, *Rec3* = *Macro3*: Pause mode. This is a reconfigurable system, i.e., it can switch the behavior from one mode to another mode. R_{skid} represents the control module of the system as:

$$\begin{aligned} R_{skid} &= Rec1 \cup Rec2 \cup Rec3 \\ &= \left\{ r_{Rec1,Rec2},\ r_{Rec1,Rec3},\ r_{Rec2,Rec1},\ r_{Rec2,Rec3}, r_{Rec3,Rec1},\ r_{Rec3,Rec2} \right\} \end{aligned}$$

For example, the second reconfiguration: "$r_{Rec1,Rec3}$" implies that "$\bullet r$" = "$Rec1$" and "$r\bullet$" = "$Rec3$". It enables to switch mode from the current to the next configuration. Figure 5 [8] helps to explain the structure of the system and the possible reconfiguration processes, i.e., it shows the possible switching mode between all the macro-steps. The *Idle* position refers to initial state where the system clock is null and the initial marking is true. It could be specified as follow:

$$\begin{aligned} I &\equiv \cap m \in Pm \\ &\equiv Idle \wedge \cap e \in Ei \neg e \wedge \cap c \in CNi \neg c \wedge (T_0 = \{1\}) \wedge (D_0 = \{0\}) \\ &\equiv Idle \end{aligned}$$

Then, according to the user choice, the system reacts to the received command. Let *EN1, EN2, EN3* be respectively the external events to activate *Rec1, Rec2, Rec3*.

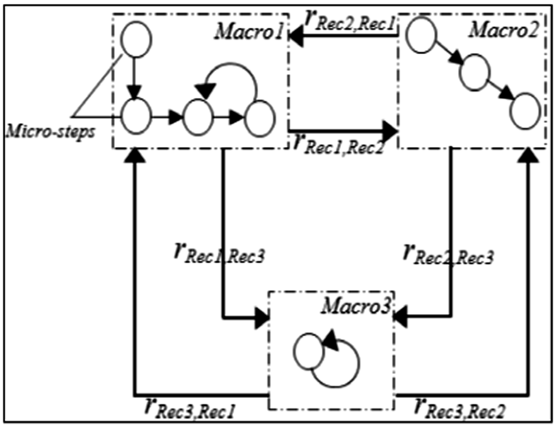


Fig. 5. System model.

Marco1 is introduced as the conjunction of the energy constraint, the memory constraint and the trigger event that will initiate the desired configuration:

$$Macro1 \equiv (E_{skid} > \text{Cost}\text{‘}Macro1\text{’}(E_0)) \wedge (Mskid > Cost\text{‘}Macro1\text{’}(M_0)) \wedge EN1.$$

The system keeps the same running mode till it receives a trigger event from the user to change the operational mode. The second reconfiguration is also introduced as follow:

$$Macro2 \equiv (E_{skid} > \text{Cost}\text{‘}Macro2\text{’}(E_0)) \wedge (M_{skid} > \text{Cost}\text{‘}Macro2\text{’}(M_0)) \wedge EN_2.$$

The system could move for the third configuration once its conjunctions are validated. This configuration is introduced as followed:

$$Macro3 \equiv (E_{skid} > \text{Cost}\text{‘}Macro3\text{’}\,(E_0)) \wedge (M_{skid} > \text{Cost}\text{‘}Macro2\text{’}(M_0)) \wedge EN_3.$$

Once the configuration is selected, the system executes the different internal tasks of the concerned macro-step. The behavior module of RTN_{skid} is formally presented as follows: $B_{skid} = (P, T, F, QW, CN, EN, DC, V, Z_0)$ where the network structure of the system is listed as: TN_{Maco1}, TN_{Maco2}, $TN_{Maco3} \in TN_{skid}$.

We have $P = P_1 \cup P_2 \cup P_3$, $T = T_1 \cup T_2 \cup T_3$, $F = F_1 \cup F_2 \cup F_3$, $W = W_1 \cup W_2 \cup W_3$, $CN = CN_1 \cup CN_2 \cup CN_3$, $EN = EN_1 \cup EN_2 \cup EN_3$, $DC = DC_1 \cup DC_2 \cup DC_3$, $V(t) = V_1(t) \cup V_2(t) \cup V_3(t)$, and $\forall p \in P_1 \cap P_2 \cap P_3$, $Z_0(p) = z_{01}(p) = z_{02}(p) = z_{03}(p)$.

We focus on the behavioral module for the specification of the system requirements, and then we aim to introduce the micro-steps of the first macro-step. The authors have to identify some properties of the system, e.g., ‘it should be possible to localize the chassis on every part of the conveyor’. Let $curr(t)$ be the system state that describes the position of chassis and $Pos1_{enb}$ be the micro-step that represent the car position in the first part of the skid. In case that the chassis should be in the first position at a prefixed time period $[a_1, b_1]$, the trigger events $E_{1.1}$ and $E_{1.2}$ should be detected at the same period. We can formally introduce this micro-state as:

$$Pos1_{enb} \equiv curr(t) \wedge E_{1.1} \wedge E_{1.2} \wedge time[a_1, b_1]$$

which evaluates the transition, i.e., it could be enabled only if all the listed conjunctions are validated. For the second position of the skid, it is formalized based on the same rules as follow:

$$Pos2_{enb} \equiv curr(t) \wedge E_{2.1} \wedge E_{2.2} \wedge time[a_2, b_2]$$

Where (i) $E_{2.1}$ and $E_{2.2}$ are the corresponding events to detect that position, and (ii) $[a_2, b_2]$ is the time period for this scenario. The proposed system aims to save the energy consumption of the transport system, i.e., the corresponding motor for each conveyor part should be *Off* if there is no car. Let $m_{1act}(t)$ be the active state of the first motor and $m_{2act}(t)$ for the second motor. Here we define the rules for $m_{2act}(t)$ as:

$$m_{2act}(t) \equiv m_{1act}(t) \wedge curr(t) \wedge E_{2.1} \wedge E_{1.2}$$

which represent that the active state of the second motor is a conjunction of the active state of the first motor, the presence of the chassis in the conveyor, the occurrence of the $E_{1.2}$: (chassis at the end of conveyor 1) and $E_{2.1}$: (chassis at the beginning of conveyor 2). To optimize the energy consumption, the system has to switch *ON/Off* the motors according to the position of the chassis. Once the second motor is turned *ON*, the first should be turned *Off*. We formalize the switching rules.

$$\neg m_{1act}(t) \equiv m_{2act}(t) \wedge \neg curr(t) \wedge E_{2.1} \wedge \neg E_{1.2}$$

which represent that stopping the first motor is initiated by the activation of the second motor, the absence of the chassis in that position is confirmed by the non-occurrence of the event $E_{1.2}$ and the occurrence of the event $E_{2.1}$. The enabled transitions can generate many events for the synchronization of the system parts. Then, the micro event $E_{2.1}$ is delivered after the movement of the chassis from the first skid to second one $(curr(t) \rightarrow curr'_{enb}(t))$. The formalization is as follow:

$$micro_e \equiv \left(\cup\, t/e \in EN_1(t) curr'_{enb}(t) \right) \leftrightarrow E_{2.1}$$

The authors present the specification of the proposed case study requirements using the presented approach. We move to the next step which is the modeling, simulation and the implementation of the system.

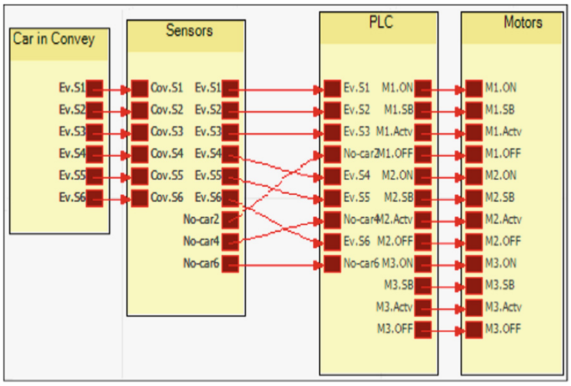


Fig. 6. Transport system model.

4.3 System Modeling

In this Section, we present the automotive transport system model based on GR-TNCES formalism using the environment ZIZO[1] [10] which is developed as a collaboration between Saarland University and Carthage University. It helps with the modeling and simulation of distributed control systems. For the purpose of optimizing the energy consumption of the old system, we present the requirements of the proposed transport system model. Based on an active control strategy of the chassis position in the skid, the new model uses additional sensors to detect the position of the workpiece on the conveyor. Figure 6 introduces the ZIZO model as a distributed discrete event system formed by four modules, e.g., the car in the conveyor, the sensors, the Programmable Logic Controller (PLC) and the three motors. Once the sensitive sensor detects the chassis in the conveyor, it sends an event signal to the PLC to activate and deactivate the corresponding motors based on the car position. The first module includes six events corresponding to the six sensors installed in the three conveyor parts. For the sensors module, it receives the events delivered by conveyor and transfers them to the PLC. It has three extra-events designed by "*No-Car2*", "*No-Car4*", and "*No-Car6*" corresponding to the events received from sensors. The third module corresponds to the PLC which controls the whole system, i.e., it receives signals from the sensors to control the state of the motors (*active, standby, off*). The events "*M1. ON*", "*M1.SB*", "*M1.Act*", and "*M1.Off*" are responsible respectively to drive the states of the motors "*Start*, *Standby*, *Active* and *Off*".

4.4 Simulation

To evaluate the energy optimization of the proposed model, we refer to the old system, i.e., we compare the consumed energy by the different models. Supposing that the motor needs four energy units (tokens) per second in the "*Active*" mode, one token in

[1] www.aut.uni-saarland.de/forschung/forschung-zizo-tool-khlifi/.

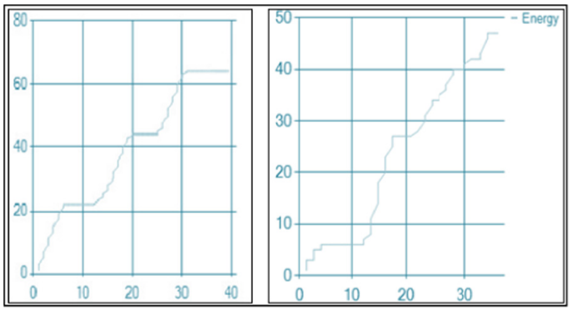


Fig. 7. Comparison of the consumed energy.

the "*Standby*" mode and zero unit if it is *Off*. Figure 7 shows the energy consumption curves during a simulation time (30 s), i.e., it presents the variation of the systems' consumed tokens. The curves corresponding to the consumption of the proposed energy efficient mode are on the right graph and the old model's consumption is on the left graph. The proposed efficient mode aims to reduce the number of active motors simultaneously, i.e., the idea is based on the detection of the car position to activate and deactivate the corresponding motors. The consumption curves present three stable parts which correspond to the deactivation period of the motors in the old model and the S*tandby* mode in the proposed model. We notice an important diminution of the consumed energy by the proposed model, i.e., in the first part (2–4 s), the consumption is highly optimized (21 to 7 tokens) since only one motor is active compared to three motors in the old model. To move the car to the second position (16–19 s), the proposed model consumes 26 tokens; on the other hand, the basic model consumes 44 energy units for the same task which is considered as a valuable optimization. Indeed, the sensors detect the car position and the PLC controls the activation and deactivation of the motors, i.e., it deactivates the first motor and activates on the second one. For the third part of the system, we succeed to save 22 energy units compared to the basic plant model.

5 Test Case: Implementation

After the successful specification, modeling, and simulation part, the conveyor system is created at Zentrum für Mechatronik und Automatisierungstechnik GmbH in Saarbrücken. As already shown, skid conveyors are transport systems that are widely used in the automotive industry. The skid conveyor belongs to the category of elevated continuous conveyors. Especially the beginning of the end of line area in an automotive production is one important application area. Here, the chassis moves continuously from one work station to the next and workers add for example the wiring harness or cockpit to the chassis. Advantages of this system are a good ergonomic working position and a good accessibility. In Sect. 4, the functional requirements and the control of the system is already described. This section shows the mechanical properties, the control system, and the implementation of these requirements.

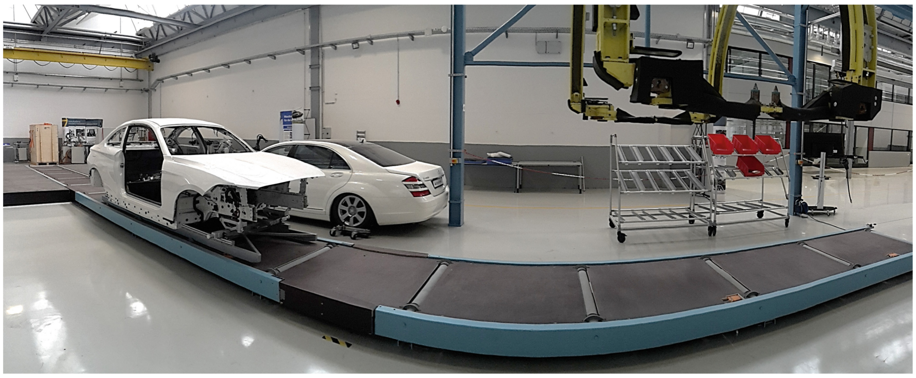


Fig. 8. Skid conveyor system at ZeMA.

5.1 Mechanical Design

As you can see in Fig. 8, the system is modular designed, i.e., it consists of three parts each equipped with one motor. This motor drives a belt with five rollers that transport a skid with the chassis on it. In order to realize energy efficient operations the system is extended by a new control system and inductive sensors. Inductive sensors generate a magnetic field that changes according to the material in immediate proximity to the sensor. With the help of these field sensors, it is possible to localize the chassis position on the conveyor. The following part shows the mechanical details of common skid conveyor systems [18]:

- Payload: up to 5500 kg
- Conveying speed: up to 2 m/s
- Gradient: up to 3°
- Web width: 1000 mm
- Load per roller: 400–800 kg.

5.2 Control System Design

The system is controlled by the interaction of different components of Fig. 9. A Programmable Logic Controller (PLC) forms the central unit of the control system and connects all sensors of the system. The PLC program, that is executed repeatedly, controls the process and communicates via PROFINET with the drives and the mobile panel. The drive system is a modular system that ranges from the control unit and power modules to the motors. User handling is realized with a mobile panel, i.e., this panel is programmed and then the user can operate the system via touchscreen.

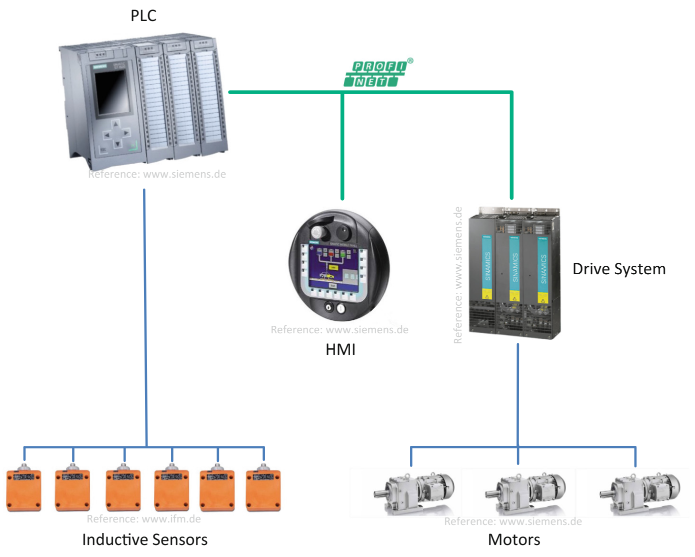

PLC
PROFI NET
Reference: www.siemens.de
Reference: www.siemens.de
HMI
Reference: www.siemens.de
Drive System
Reference: www.ifm.de
Inductive Sensors
Reference: www.siemens.de
Motors

Fig. 9. Design of the control system.

5.3 Software Implementation

In order to control the skid conveyor system, the PLC has to be programmed with the desired functionality. The international standard IEC 61131 [19] represents the standard for programmable logic controllers and according to the standard; there are several possible programming languages. In our case, we used the programming languages structured text (ST) and function block diagram (FBD) to realize the desired control behaviour. For its usefulness, a graphical user interface is implemented in the panel; therefore, we made use of a process visualization system. In this way, the user can select one of the three following user modes that are implemented in the PLC:

- Automatic mode,
- Manual mode, and
- Pause mode,

We strictly concentrate on the description of Sect. 4, i.e., we implemented the functionality in the PLC program with the mentioned programming languages. In the requirements is also explained that we can switch the components into energy saving modes in the pause mode for example. This fact is realized with the help of the PROFIenergy profile which is a PROFINET [20] based profile that makes it possible to set certain components into energy efficient modes as soon as the pause mode is activated. In this case, the PLC sends a command to the components to start and end the *standby* mode.

6 Discussion

We proposed a new specification approach that is much more expressive and optimized compared to statecharts used in symbolic model checking. The proposed approach has ability to cope with reconfigurable systems and timed constraints which is not possible in statecharts. It is possible to express more restrictions related to timed systems and real time process and enables to describe systems that could change their behavior recognized as adaptive systems. It is also possible to use deferent firing mode for the system transition states: i.e., we can opt from *AND/OR* mode according to the system requirements. ("*AND*" if all the transition inputs are required and "*OR*" is used if one them could validate the transition). Thanks to this approach, it is possible to check the availability of resources before starting such a reconfiguration process, i.e., to guarantee the non-resources violation once the system executes its tasks. Unpredictable behaviors are also covered here, i.e., the specification approach is able to describe the probabilistic behavior. We present also a complete approach ranging from specification, modeling, and simulation to the real implementation of the proposed automotive transport system. The ZIZO model proofs that the proposed model saves the energy consumed by the transport system compared with the old version of the system.

7 Conclusion

This paper proposes a new specification approach dealing with unpredictable flexible control systems running under memory and energy resources constraints. It is an expressive method that could specify limited memory and energy reserves, probabilistic behaviours, and reconfigurable processes which was not discussed in the previous work. The proposed approach is based on GR-TNCES formalism which enables us to express the probabilistic reconfiguration scenario of such a system. An automotive transport system is the considered case study to concretize the contribution. We present therefore the specification, modelling, simulation, and implementation of the transport system. The reconfigurations and the functionalities of the system are specified thanks to this approach. A new GR-TNCES model with aim to save energy is developed and simulated using ZIZO to evaluate its energy consumption compared to old system model. The reported results of the improved system show an important reduction of the consumed energy, i.e., more than 60% in the first part of the conveyor and more than 40% in the second part. The implementation of the real transport system is also presented in this work. During the next step of this project, we will work on the validation of the proposed model through a real energy data measurement of the skid conveyor.

Acknowledgement. This work was supported by Zentrum für Mechatronik und Automatisierungstechnik" (ZeMA) and "ERASMUS + Program" at Saarland University – promoted by the European Program. This research work is a collaboration between Saarland University, Germany and University of Carthage, Tunisia.

References

1. Khlifi, O., Mosbahi, O., Khalgui, M., Frey, G.: GR-TNCES: new extensions of R-TNCES for modeling and verification of flexible systems under energy and memory constraints. In: International Conference on Software Technologies, Colmar, France, pp. 373–380 (2015)
2. Bortolussi. L., et al.: Verification of Complex Adaptive Systems (2015). http://homepage.lnu.se/staff/daweaa/papers/2015CASVerification.pdf
3. Zhang, J., Khalgui, M., Li, Z.W., Mosbahi, O., Al-Ahmari, A.M.: R-TNCES: a novel formalism for reconfigurable discrete event control systems. IEEE Trans. Syst. Man Cybern. Syst. **43**(4), 757–772 (2013)
4. Harel, D., et al.: STATEMATE: a working environment for the development of complex reactive systems. IEEE Trans. Softw. Eng. **16**(4), 403–414 (1990)
5. Leveson, N.G., Heimdahl, M.P.E., Hildreth, H., Reese, J.D.: Requirements specification for process-control systems. IEEE Trans. Softw. Eng. **20**(9), 684–707 (1994)
6. Bastide, R., Buchs, D.: Models, formalisms and methods for object-oriented distributed computing. In: Bosch, J., Mitchell, S. (eds.) ECOOP 1997. LNCS, vol. 1357, pp. 221–255. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-69687-3_45
7. Chan, W., et al.: Optimizing symbolic model checking for statecharts. IEEE Trans. Softw. Eng. **27**(2), 170–190 (2001)
8. Khlifi, O., Siegwart, C., Mosbahi, O., Khalgui, M., Frey, G.: Specification approach using GR-TNCES -application to an automotive transport system. In: 12th International Conference on Software Technologies, Madrid, Spain (2017)
9. Andrade, E., Maciel, P., Callou, G., Nogueira, B.: A methodology for mapping SysML activity diagram to time petri net for requirement validation of embedded real-time systems with energy constraints. In: 3rd International Conference on Digital Society, Cancun, Mexico, pp. 266–271 (2009)
10. Salem, M.O.B., Mosbahi, O., Khalgui, M., Frey, G.: ZiZo: modeling, simulation and verification of reconfigurable real-time control tasks sharing adaptive resources: application to the medical project BROS. In: International Conference on Health Informatics, Portugal, pp. 20–31 (2015)
11. Chen, Y.F., Li, Z.W., Zhou, M.C.: Optimal supervisory control of flexible manufacturing systems by petri nets: a set classification approach. IEEE Trans. Autom. Sci. Eng. **11**(2), 549–563 (2014)
12. Harel, D.: Statecharts: a visual formalism for complex systems. Sci. Comput. Program. **8**(3), 231–274 (1987)
13. Wasserman, A.: Extending state transition diagrams for the specification of human-computer interaction. IEEE Trans. Softw. Eng. **11**(8), 699–713 (1985)
14. Ross, D.: Structured analysis (SA): a language for communicating ideas. IEEE Trans. Softw. Eng. **SE-3**(1), 16–34 (1997)
15. Zedan, H., Cau, A., Chen, Z.: Yang. H.: ATOM: an object-based formal method for real-time systems. Ann. Softw. Eng. **7**, 235–256 (1999)
16. El-kustaban, A., Moszkowski, B., Cau, A.: Specification analysis of transactional memory using ITL and AnaTempura. In: Lecture Notes in Engineering and Computer Science, pp. 176–181 (2012)
17. Khlifi, O., Siegwart, C., Mosbahi, O., Khalgui, M., Frey, G.: Modeling and simulation of an energy efficient skid conveyor using ZIZO. In: 13th International Conference on Informatics in Control, Automation and Robotics (ICINCO), Lisbon, Portugal, pp. 551–558 (2016). ISBN 978-989-758-198-4

18. Hompel, M., Schmidt, T., Nagel, L.: Materialflusssysteme: Förder und Lagertechnik (Material Flow Systems: Conveyor and Storage Techniques). Springer, Berlin (2007). https://doi.org/10.1007/978-3-540-73236-5. ISBN 978-3-540-73235-8
19. IEC 61131-3: Programmable controllers – part 3: programming languages, International Standard, International Electrotechnical Commission (2013)
20. PROFIBUS Nutzerorganisation e.V., "Pi White Paper: The PROFIenergy Profile," Karlsruhe, Germany, pp. 10–11 (2010)

Towards a Goal-Oriented Framework for Partial Agile Adoption

Soreangsey Kiv[1(✉)], Samedi Heng[1], Yves Wautelet[2], and Manuel Kolp[1]

[1] Louvain Research Institute in Management and Organizations (LouRIM), Université Catholique de Louvain, Ottignies-Louvain-la-Neuve, Belgium
{soreansey.kiv,samedi.heng,manuel.kolp}@uclouvain.be
[2] Faculty of Economics and Business (FEB), Leuven, KU, Belgium
yves.wautelet@kuleuven.be

Abstract. The agile paradigm is used today for software development and project as an alternative to structured and traditional heavier life cycles. Different meta-models have been proposed trying to unify agile methods. Yet, very few of them focus on agile partial method adoption. Intuitively, choosing which practices to adopt from agile methods should be made based on their most prioritized goals in the software development process. The paper answers this issue by building a goal-oriented meta-model where each agile concept is seen as a goal to achieve and explaining how goal modeling can help the software team to partially adopt agile methods. This will also make it easier to identify vulnerabilities associated with each goal and minimize risks.

Keywords: Agile methods · Partial adoption · Goal modeling Meta-model · Requirements engineering

1 Introduction

Software development approaches such as eXtreme Proramming (XP) [1], Feature-Driven Development (FDD) [2], Dynamic Systems Development Method (DSDM) [3], Crystal family [4], Scrum [5] have led to the definition of the Agile Manifesto [6,7] to offer alternatives to traditional software life cycle. The manifesto defines a set of values and principles to be followed by software development processes, the most important principle being iterative life cycles (Deming/Shewart cycle, spiral, sprint, scrum, ...). Many studies demonstrated that iterative processes and phases allow change, increase quality and reduce the risk of failure. Agile and iterative processes have thus emerged as popular approaches.

Intuitively, when software teams want to partially adopt agile methods – either one or the combination of multiple methodologies – they should have in mind the reasons why they want to do it and the goals they want to achieve after the adoption. Tripp and Armstrong [8] conduct an exploratory study and point out different motives for agile adoption, all with different project management configurations focusing on agile practices for software development. Campanelli

E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 69–90, 2018.
https://doi.org/10.1007/978-3-319-93641-3_4

and Parreiras [9] highlight that 42% of the papers working on agile tailoring use goals as one of their criteria for practice selection. For instance, among others, the Strategic Analysis for Agile Practices framework [10] proposes to link the selection of agile practices to the organization's business goals.

Unfortunately, these business goals along with other criteria are often described textually. To ensure wide diffusion in academia, we believe that explicit formalisms should be proposed. By using meta-models such as [11–13], methods can be described in a more precise (and formal) way and offer more explicit representation and guidelines. This has led us to propose, by means of a meta-model, a partial agile adoption framework based on a goal perspective. Our motivation contrasts with previous goal-based agile tailoring approaches in the sense that we employ the original (software development) goals behind agile methods creation (i.e., the Agile Manifesto) rather than business goals. This perspective coincides also with the work of [14] which emphasizes that agile practices should follow the agile values which are fundamental to define the culture of the software company.

Our main objective is to build a goal-oriented meta-model where each agile concept will be seen as a goal to achieve and to which dependencies between roles and resources are associated. A complementary objective is to explain how we make use of social intentional modeling and our meta-model to help the software team to partially adopt agile methods. This paper is an extended version of [15].

The paper is organized as follows. Section 2 discusses various agile methods and meta-models proposed in literature. Their pros and cons are highlighted. Section 3 presents our framework for partial agile adoption. We first expose our agile meta-model through its tactical and operational levels. Then, we discuss the use of the intentional i* framework [16] to instantiate our meta-model for better visualization. Next, we provide a methodology for using our framework for partial agile methods adoption. Section 4 provides a validation for our approach. Finally, we conclude the paper in Sect. 5 including a discussion on future directions.

2 Related Work

Over the last decade, many agile methods have been proposed to meet specific requirements and situations. For instance, Scrum has the objective to put more focus on project management considerations while XP is designed to be more responsive to customer requirement changes [12]. [17] points out that agile methods tend to be compliant with small development teams and focus more on people rather than processes or artifacts. Other well-known instances and inspirations of agile-based frameworks can be found in the literature, some are even derived from other disciplines than software and information systems engineering: Agile Unified Process (AUP) [18], OpenUP [19] and Agile Modeling [20], Kanban [21,22], Lean Software Development [23], Lean Office and Organization [24], etc.

Agile methods are generally described textually as a series of values, principles and practices [1,5,6]. To formalize their descriptions and to make the method adoption process better and faster, and at the same time to minimize the risks, meta-models have been created in different perspectives. We overview relevant related studies on agile metamodeling below.

Schwaber [25] proposes a meta-model to support the construction of agile methods. By relying on metrics on a Situational Method Engineering framework [26], he provides guidance to agile methodologists during the construction and throughout the development process itself. Rather than working on a single agile method, Mikulėnas et al. [12] focus, on their part, on a meta-model enabling the fusion of different agile methods referred as *partial agile methods adoption*. The core idea is to decompose each agile method into components which, in theory, could always be categorized into a common pre-defined structure. This offers great flexibility in adopting agile methods since one can manually select and combine different alternative components coming from different methods at will.

Interestingly, Lin et al. [27] and Esfahani et al. [13] have shed more light on goal-oriented meta-models. Lin et al. [27] propose a novel goal-oriented method to model a software development process on the top of the AUP [18], called Goal Oriented Agile Unified Process (GOAUP) that could also be applied to OpenUP and other agile-oriented forks of the *Unified Process*. Goal-Net theory [28] is used to model the hierarchical goals in the AUP through a so-called Goal-Net diagram. In that purpose, each iteration has one main goal. Esfahani et al. [13] propose a more sophisticated goal-oriented meta-model that put focus on social interaction in which human relations are clearly defined. For example, the relationship/dependency between roles in required processes and the skills are associated with each goal in turn described with certain vulnerabilities to minimize the risks of the adoption process. Although many other meta-models have been proposed [11–13,27–31], none of them focuses specifically on partial agile adoption in an intentional goal-oriented perspective. The closest research work related to ours is [13], in which goal-oriented software process modeling has been used to model micro-processes (i.e., practices). However, the authors focus rather on how to visualize agile methods, mainly Scrum, through social modeling while our work aims at using a goal-oriented framework specifically dedicated to partial agile methods adoption.

Wautelet et al. [32] use a process fragment approach (see [33]) to include requirements models developed in previous contributions (i.e. [34–36]) as artifacts of any agile software development method. To this end, the authors instantiate the generic process fragment template of [37] and map each concept to the i* framework [38] for visual representation. Their approach is of particular interest here in the sense that they also depict a software development process with i* but the adoption of the process fragment is done on an ad-hoc basis rather than on the basis of values. Indeed, while our partial agile process selection mechanisms relies on high level values, their approach is targeted to simply include some specific software development practices within an existing agile method in an as-is fashion without any element selection process. Our approach is thus top-down while theirs is rather a bottom-up add-on to enrich the software engineering stage of the parent agile method adopting their process fragment.

Finally, let us note that their use of the i* concepts for the visual representation of the software development process is not aligned with ours; it is indeed not driven by the same ontological basis. They rely on a third-party process fragment while we here develop our custom meta-model later mapped to i*.

Last but not least, we have chosen to represent the process elements in our partial agile method adoption framework with i* even if specific frameworks for software development process representation exist. We can notably refer here to the Software & Systems Process Engineering Metamodel (SPEM) [39] (see [40] for an application on agent-oriented iterative life cycle representation), ISO/IEC 12207 [41] and ISO/IEC 15504 [42]. The major drawback of these frameworks is that they are not goal-oriented so that they do not allow tracing the rationale behind each operational level element choice. In other words, they are designed for representing a single software development method at operational level not for reasoning about a possible composition of several methods.

3 A Framework for Partial Agile Adoption

As evoked, our objective is to introduce a framework for partial agile methods adoption in a goal-oreinted and social perspective. The aim is threefold: to provide (1) a goal-oriented meta-model for describing agile methods, (2) a goal and social representation by using the i* framework and (3) an approach for practices selection and adoption.

Goal-Oriented Meta-model: This section describes how the meta-model is created, how each agile concept is seen in the goal perspective and finally, how this meta-model can be accommodated to any agile method.

Goal and Social Representation: Up to this point, agile concepts have been discussed and represented in the meta-model. This section describes how to use a social intentional modeling framework such as i* for a better visualization of socio-intentional dependencies.

Approach for Practices Selection and Adoption: This section explains how representation of goal-oriented agile concepts with i* can be useful for practices selection and how it can help practitioners to identify the possible risks.

3.1 Goal-Oriented Agile Methods Meta-model

The Agile Manifesto introduced four values and twelve principles [6] agile practitioners should respect in order to be agile compliant. Following Madeyski [43], *"agile values are the large-scale criteria we use to judge what we see, what we think and what we do"*. Most agile methods nevertheless come with their own set of practices – for instance, Scrum possesses 14 practices and XP has 13 practices (see Table 2) – in fact, these practices are the particular means to fulfill some values and principles of the Agile Manifesto. Madi et al. [14] mentioned that values of the Agile Manifesto are fundamental for any agile method because a set of practices can be followed on their basis and knowing the most important agile values is the key to follow the best set of practices. According to Sidky et al. [44], practices are concrete activities and practical techniques used to develop and manage software projects in accordance with the agile principles.

Based on both [14,44], value and principle are the key elements that define which practices to adopt. However, we can see that values are actually the abstract ideas implemented by principles. Thus, principles are considered as the bridge that narrows the gap between the abstract general values and the detailed specific practices [45].

To the best of our knowledge, relationships between agile values, principles and practices have never been explicitly stated. However, they can be literally understood from their own descriptions. For instance, one of the four agile values, – *"Individuals and interactions over processes and tools"* – can easily be related to the software team and its interactions. Basically, we can say that a principle contributes to this value if it helps to improve the ability of the software team (P1, P3, P4 and P5) and their interactions (P2)[1].

Principles of the Agile Manifesto are however still an abstraction and they focus on multiple aspects of software development. Laanti et al. [46] conducted research on decomposing each principle into fine-grained elements which they called the *emphasis* of the principle. We define the latter concept in our meta-model as *Agile Feature*. It allows better mapping practices with principles. In other words, an agile feature allows filling the gap between principles and practices. For example, the principle P1, *"build project around motivated individuals. Give them the environment and support they need, and trust them to get the job done"*, means that one should emphasize on *motivation of individual*, *working environment*, *support* and *trust*. The latter are referred to as "agile features" in our meta-model. To achieve this principle, if practitioners use Scrum and XP methods, the suitable practices are[2]:

- Motivated individuals: this can be fulfilled using practices "Daily meeting" (SP4 or XP4), "Three questions" (SP5) and "Sign up" (XP3). "Daily meeting" and "Three questions" help to motivate members by asking them to report everyday on the problems they face and the help they need to solve them. "Sign up" practice helps to improve the individual's motivation by encouraging them to volunteer in choosing the task to perform;
- Good environment: this can be fulfilled by using practices "Daily meeting" (SP4 or XP4), "Three questions" (SP5) and "Sustainable pace" (XP1). "Daily meeting" and "Three questions" can help to improve the working environment by knowing what bothers each individual and removing the problem as quickly

[1] Principles are denoted as $Px, x \in \{1 \ldots 12\}$ as seen in Table 3.

[2] Scrum practices are denoted as $SPx, x \in \{1 \ldots 14\}$ and XP practices are denoted as $XPx, x \in \{1 \ldots 13\}$ as seen in Table 3.

as possible. "Sustainable pace" helps planning releases and at the same time keeps the team from getting into a death march;
- Support: this can be fulfilled by using practice "Pair programming" (XP2). It is a kind of technique where two people work together using a single computer. This kind of technique will improve the quality of the code and at the same time make individuals feel supported, for example when the partner says "let's try your idea first";
- Trust: this can be fulfilled by using practices "Sign up" (XP3) and "Collective ownership" (XP9). These two practices can only be adopted when the team trusts each other in choosing the task to do.

By understanding the textual meaning, we can always find which principles contribute to which values, which agile features emphasize which principles and which practices can help to achieve which agile features.

Finally, in order to adopt the relevant practices, we need certain roles to perform, with/without using/producing the artifact(s).

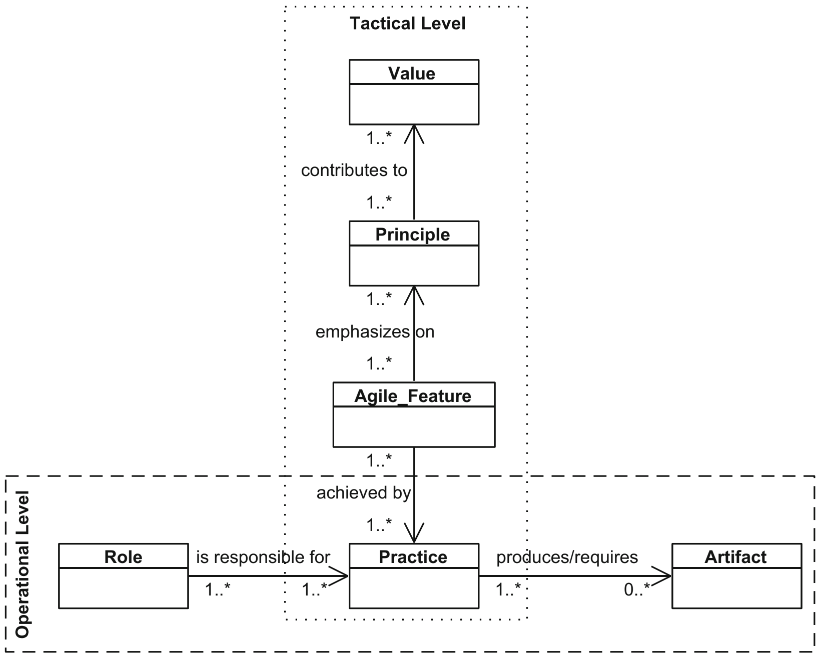


Fig. 1. Goal-oriented agile meta-model [15].

To sum up, our meta-model is built on top of these abstractions: *value*, *principle*, *agile feature*, *practice*, *role* and *artifact*. Figure 1 exposes the meta-model of agile methods in a goal-oriented perspective, the main components are:

- *Value* is the large-scale criteria used to judge what we see, what we think and what we do [43]. It is seen as the top-level goal in agile methods adoption;
- *Principle* helps to establish a mind-set for solid software engineering practices [47]. It comes into agile methods and acts as a bridge to narrow the gap between values and practices [43];
- *Agile feature* is the distinctive characteristic of each principle, as defined by [46]. These agile features can help to narrow the gap between principles and practices, and consequently the relevant practices can be better identified by the development team;
- *Practice* is the concrete activity and practical technique that is used to develop and manage software projects in a manner consistent with the agile principles [44];
- *Role* in agile methods refers to the allocation of specific roles through which the software production in a development team is carried out [44];
- *Artifact* is the resource produced during software development. It can be tangible like user stories [34,48] or intangible like working software functionality.

Modern agile methods have usually been proposed with their own set of values, principles, practices, roles and artifacts. Redundancy or overlapping between these elements from one method to another is inevitable and elements may be named differently in different methods, yet they are referring to the same idea. Thus, we strongly believe that a principle can contribute to many values (having the same goal), one agile feature can emphasize on many principles (having the same meaning) and so on. Hence, all relationships between agile concepts in our meta-model are of the type "many-to-many".

3.2 Goal and Social Dependency Representation

In order to visualize the instance of the elements from our meta-model as a goal perspective, we use the Strategy Dependency (SD) model. The analysis based on graphical representation allows us to see which practices can fulfill which goals (i.e., values, principles and agile features) and identify the relevant practices to fulfill a selected goal. Moreover, the SD model allows us to consider the social dependency between roles involved in the software development team for some practices. This allows the team to identify the vulnerabilities of adopting a particular practice.

We briefly describe hereafter the related concepts of the i* framework and explain how we map elements in our meta-model with elements in i*.

The i* Modeling Framework. Goal/Intention and social dependencies between the various components over a specific goal should be visualized using goal-oriented modeling frameworks. As already said, in this paper, we use the i* framework because of its popularity and its social and intentional modeling possibilities. A full description of the i* concepts can be found in [49] but we summarize the important ones below for self-sufficiency:

- *Hard-goal* is an intentional desire of an actor. The specific way of how the goal is to be satisfied is not described;
- *Soft-goal* is similar to (hard) goals except that the criteria for the goal's satisfaction are not clear-cut. It is up to the actor to judge;
- *Task* is a particular way of attaining a goal;
- *Resource* is the finished product of some deliberation-action process;
- *Role* is an abstract characterization of the behavior of a social actor within some specialized context or domain of endeavor. Its characteristics are easily transferable to other social actors;
- *Agent* is an actor with concrete, physical manifestations, such as a human individual. An agent has dependencies that apply regardless of what roles he/she/it happens to be playing;
- *Contribution link (some+)* is a positive contribution whose strength is unknown;
- *Contribution link (and)* is a kind of contribution where the parent is satisfied if all of the offsprings are satisfied;
- *Contribution link (make)* is a positive contribution strong enough to satisfy a soft-goal;
- *Task dependency* is a relationship where the depender depends on the dependee to carry out an activity (task);
- *Resource dependency* is a relationship where the depender depends on the dependee for the availability of an entity (physical or informational).

These elements are used in two main models: the Strategic Dependency model (SD) and Strategic Rationale model (SR). The SD model describes external dependency relationships between goals, actors using hard-goal, soft-goal, task and resource elements. These four kinds of dependencies are suitable to represent respectively the functional goal, the functional goal without a clear-cut of satisfiability, the functional goal with a specific activity or practice to achieve and the resource or artifact needed during the development. In addition to the SD, the SR model provides a deeper representation of the internal, intentional dependency with the means-end, task-decomposition and contribution links to describe the stakeholders' interests and the (combination of) activities, sub-goals, artifacts that help to achieve the main goal.

Mapping Our Agile Meta-model Concepts with i*. The mapping is performed with a heuristic search using the idea of Cartesian product. Indeed, for every concept in our meta-model, we compare its definition/objective to all the possible i* elements that can be found. The best match should have the closest objective/definition. We first map the classes and finish with mapping their relationships. We managed to map all the concepts and their relationships with i* elements and detail hereafter the motivation of our mapping for each concept (Table 1):

Table 1. Mapping agile concepts with i* [15].

Agile concepts	i* concepts
Value	Soft-goal
Principle	Soft-goal
Agile feature	Soft-goal
Practice	Task
Artifact	Resource
Role	Role or agent
Contributes to	Contribution link (some+)
Emphasizes on	Contribution link (and or make)
Achieved by	Contribution link
Is responsible for	Task dependency
Produces/requires	Resource dependency

- *Value*, *principle* and *agile feature* are represented as Soft-goal. These concepts are described in agile methods as the objectives to be achieved without any defined criterion. For instance, *"Individuals and interactions over processes and tools"* explains why software team members and their interactions are important for the development but there is no explicit explanation of how the team should be structured nor what kind of interactions they need in order to attain the benefit of this value. Similar explanations apply for the concepts of principle and agile feature;
- *Practice* is described as a set of activities that the software team must follow in order to realize its benefit. It is represented as a Task in i*. Within a practice, there are tasks to perform and they can require effort from many roles in the whole team. For example, the practice "Daily meeting" requires "Scrum Master" or "XP Coach" to organize the meeting and requires the participation of an "Agile team". However, we have chosen not to discuss about tasks in this paper for several reasons. First, the term *task* has never been explained as the core concept in agile methods. Second, there is no clearcut on how we can define *task* within a practice. In order to do so, we need a long and well defined literature review. This is beyond the objective of this paper which is a preliminary study of the agile partial adoption in the goal perspective;
- *Artifact* is the resource required or created when performing a practice. For example "User stories" are the resources needed for a "Sprint planning" and from this practice, a "Sprint backlog" is created. In i*, an artifact is represented as a resource;
- *Role* exists in many forms in different agile methods. Each role has different responsibilities and can be played by different or the same actor. It is represented as a Role or Agent in the i* framework;

- *Contributes to* is the relationship between value and principle which are soft-goals. It is represented as a contribution link with the "some+" tag, as the strength of its contribution to satisfy is unknown;
- *Emphasizes on* is the relationship between principle and agile feature. Each principle is decomposed into different parts used to emphasize it. This kind of relationship is represented by a contribution link with the tag "and" while principle is the parent and can be fully satisfied only if all of the children, agile features, are satisfied. Some principles having only one feature, the "contribution link" with the tag "make" is then used to represent them. That one feature is the only thing to be satisfied to fulfill the principle it emphasizes;
- *Achieved by* is the relationship between agile feature (soft-goal) and practice (task). This relationship is a "contribution link" where the tag used depends on the practice to adopt and the feature to achieve;
- *Is responsible for* is the relationship between an agile role and the practice under its responsibility. It is represented by task dependency;
- *Produces/requires* is the relationship between an agile role and the artifact it requires or creates. It is represented by an i* resource dependency, which roles depend on each other for the resource needed.

3.3 Partial Agile Adoption Process

Partial agile methods adoption aims at choosing the right practices and integrating them successfully into software development process. To do so, the development team should use one or a combination of multiple methodologies to specify the goals to be achieved by the end of the adoption process. Thus, in our goal-based framework, the adoption process starts from defining the goals, followed by other steps to be performed in an iterative and incremental[3] way. We divide these steps into two levels: *tactical* and *operational levels.*

- At the tactical level, the software team defines goals to be achieved with the adoption of the agile methods. Then the team selects related practices by using the meta-model presented in the previous section;
- At the operational level, the software team identifies possible risks to manage in order to successfully adopt agile practices.

Figure 2 illustrates the process of partial agile methods adoption using our framework.

The steps of the partial agile methods adoption process are:

- *Defining Goals*: the software development team defines the goals/objectives they want to achieve after adopting the agile methods. The goals here refer to the ones the development team adheres to and not the business goals they aim to satisfy with the software development. In our research, we refer to the goals behind the agile methods creation, i.e., *values*, *principles* and *agile features*;

[3] One iteration per method candidate to be partially adapted.

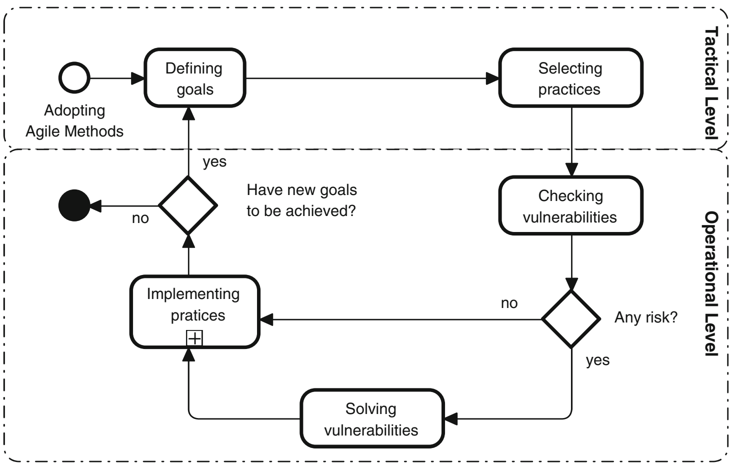


Fig. 2. Goal-oriented partial agile adoption process.

- *Selecting Practices*: as seen in our meta-model, a practice is linked to an agile feature. So, based on the previously defined goals (value, principle and agile feature), using the i* representation at this step allows the teams to identify practices relevant to fulfill those goals. The team can thus select a set of practices to be adopted. Section 4.2 gives details and an example;
- *Checking Vulnerabilities*: the team cannot successfully integrate agile practices into the development process unless the role that is in charge of these practices performs correctly and artifacts consumed by practices are sufficiently provided. That is why, in our framework, various relationships/dependencies between agile practices, roles, and resources are visualized using the SD model. This will enable the team to see the possible vulnerabilities. Indeed, when a role is unable to perform correctly, the required practices or artifacts cannot be provided. Extra information on how to build dependencies and visualize them in the SD diagram can be found in Sect. 4.3;
- *Solving Vulnerabilities*: based on the results of the dependency checks, if any risk is identified and associated with the practices, the team needs to solve the cause of vulnerability by reinforcing the roles and artifacts needed;
- *Implementing Practices*: If there is not any risk, the team can directly start implementing the practices into their development process.

4 Validation

In order to fully validate the framework, two necessary steps should be carried out: *theoretical* and *practical* validation. The *theoretical* validation aims at verifying that all the agile concepts can always be mapped to our meta-model and showing that it can help a practitioner in selecting practices and identifying vulnerabilities. The *practical* validation aims at validating whether or not the framework is useful for the development team in the real case of partial agile methods adoption. However, as a preliminary study, we only focus on the theoretical validation in this paper.

A significant number of agile practices are available within the agile community. The Agile Alliance [50] has documented some of the well-known ones. Nevertheless, we cannot consider all agile practices to discuss them in this paper. We thus have chosen to work only with the two most popular agile methods, namely Scrum and XP. Concretely, the *theoretical* validation is conducted at two levels, i.e., the tactical and operational ones. Firstly, at the tactical level, the relationships between *value*, *principle*, *agile feature* and *practice* will be discussed and visualized. Practitioners can follow our methodology to find out which practices fulfill their desired goals. Secondly, at the operational level, the dependencies between selected practices, roles and artifacts will be described and visualized by using an SD diagram. This step helps practitioners to identify the vulnerabilities of adopting those practices. We briefly describe Scrum and XP methods in the following section and describe validations at the tactical and operational levels.

4.1 Scrum and XP

Scrum, the most commonly used agile method [51], is a framework within which people can address flexible adaptive problems, while productively and creatively delivering products of the highest possible value [52]. It is neither a process nor a technique for building products. It is rather a method within which we can employ various processes and techniques. It consists of Scrum Teams and their associated roles, events, artifacts, and rules. Each component within the framework serves a specific purpose and is essential to the success and usage of Scrum. The rules of Scrum bind together the events, roles, and artifacts, governing the relationships and interaction between them. Scrum has never been described with clear-cut practices. Based on the Agile Alliance [50], the Scrum practices are exposed in Table 2.

As described in [1], ***XP*** is a process for the business of software development that makes the whole software team focus on common and reachable goals. Using XP values and principles, software teams apply appropriate practices in their own context. XP practices are chosen for their encouragement of human creativity and their acceptance of human frailty. One of the goals of XP is to bring accountability and transparency to software development to run software development like any other business activity. Another goal is to achieve more effective and efficient development with far fewer defects than is currently expected. Finally, XP aims to achieve these goals by celebrating and serving the human needs of everyone touched by software development sponsors, managers, testers, users, and programmers. Based on the Agile Alliance [50], XP practices are exposed in Table 2.

Table 2. Scrum and XP practices.

Scrum practices	XP practices
SP1: Iterative development, **SP2:** Timebox, **SP3:** Iteration, **SP4:** Daily meeting, **SP5:** Three questions, **SP6:** Burndown chart, **SP7:** Task board, **SP8:** Definition of done, **SP9:** Definition of ready, **SP10:** Point estimates, **SP11:** Relative estimates, **SP12:** Planning poker, **SP13:** Backlog, **SP14:** Backlog grooming	**XP1:** Sustainable pace, **XP2:** Pair programming, **XP3:** Sign up, **XP4:** Daily meeting, **XP5:** Iteration, **XP6:** Velocity, **XP7:** Frequent release, **XP8:** User stories, **XP9:** Collective ownership, **XP10:** Continuous integration, **XP11:** Simple design, **XP12:** Refactoring, **XP13:** Test Driven Development (TDD)

4.2 Tactical Level Application: Towards Practices Selection

Validation at the tactical level is performed in two steps. The first one consists in showing that values, principles, agile features and practices really contribute to one another. The second step aims at explaining how the representation in i* helps agile practitioners to find the relevant practices supporting their goals. Both steps are described hereafter.

Mapping Agile Values, Principles, Features and Practices. As mentioned, relationships between the specific agile values, principles and practices have never been explicitly stated but can be understood from their descriptions. With many agile methods proposed over the years, a lot of agile values, principles and practices can be used to validate the idea.

We took the four values and twelve principles of the Agile Manifesto, twenty three agile features defined in [46] and twenty seven practices from Scrum and XP to analyze and build the relationships between one another. The mapping is done empirically based on our understanding and interpretation. However, we carried out a double check with senior researchers and professors. The result is summarized in Table 3.

As a result, each value is contributed by at least one principle, each principle is emphasized by at least one agile feature and each agile feature can be achieved by at least one practice. Inversely, all practices can help to achieve at least one agile feature, each agile feature emphasizes one principle and each principle contributes to one value.

Nevertheless, this validation does not involve agile features that emphasize on multiple principles, and none of the principles contributes to multiple values. However, we strongly believe that, with many values and principles from different agile methods, their relationships of emphasis and contribution will become multiple.

Table 3. Mapping agile values, principles, features and practices (Scrum and XP).

Value	Principle	Agile feature	Agile practices
V1: Individuals and interactions over processes and tools	**P1:** Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done	**AF1:** Motivated individuals	SP4, SP5, XP3, XP4
		AF2: Good environment	SP4, SP5, XP1, XP4
		AF3: Support	XP2
		AF4: Trust	XP3, XP9
	P2: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation	**AF5:** Efficiency (for conveying information)	SP12, SP13, SP14, XP8
		AF6: Communication	SP4, SP5, SP12, SP14, XP2, XP4
	P3: Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely	**AF7:** Sustainability	SP1, SP2, SP3, SP6, SP7, SP10, SP11, XP1, XP5, XP6, XP7, XP10
		AF8: People	SP5, SP4, XP2, XP3, XP4
	P4: The best architectures, requirements, and designs emerge from self-organizing teams	**AF9:** Self-organization	SP4, SP12, XP2, XP3, XP4, XP9
	P5: At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly	**AF10:** Built-in improvement of efficiency and behavior	SP4, SP5, XP4
V2: Working software over comprehensive documentation	**P6:** Our highest priority is to satisfy the customer through early and continuous delivery of valuable software	**AF11:** Customer satisfaction	SP8, SP9, SP12, SP13, SP14, XP8, XP13
		AF12: Continuous delivery	SP2, SP1, SP3, XP5, XP10
		AF13: Value	SP13, XP8
		AF14: Early deliveries	XP11, XP12
	P7: Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale	**AF15:** Frequent deliveries	SP1, SP2, SP3, XP5, XP7, XP10, XP11, XP12
	P8: Working software is the primary measure of progress	**AF16:** Measure progress via deliverables	SP6, SP10, SP11, XP6
	P9: Simplicity—the art of maximizing the amount of work not done—is essential	**AF17:** Simplicity	SP1, SP3, XP5, XP7, XP10, XP11
		AF18: Optimize work	XP12
V3: Customer collaboration over contract negotiation	**P10:** Business people and developers must work together daily throughout the project	**AF19:** Collaboration	SP4, SP5, SP12, SP14, XP2, XP3, XP4, XP9
V4: Responding to change over following a plan	**P11:** Welcome changing requirements, even late in development	**AF20:** Adaptability	SP1, SP3, XP5, XP7, XP10, XP11
		AF21: Competitiveness	SP1, SP3, XP5, XP7, XP10, XP11
		AF22: Customer benefit	SP12, SP13, SP14
	P12: Continuous attention to technical excellence and good design enhances agility	**AF23:** Focus on technical excellence	SP1, SP3, XP2, XP5, XP9, XP10, XP13

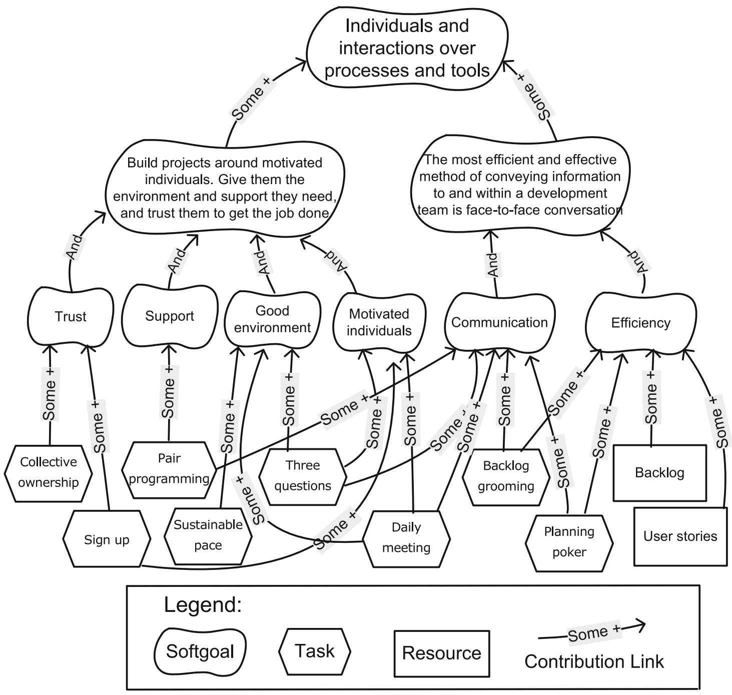

Individuals and interactions over processes and tools
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
Trust
Support
Good environment
Motivated individuals
Communication
Efficiency
Collective ownership
Pair programming
Three questions
Backlog grooming
Backlog
Sign up
Sustainable pace
Daily meeting
Planning poker
User stories
Some +
Some -
And
Legend:
Softgoal
Task
Resource
Contribution Link

Fig. 3. Relationship between values, principles, features and practices at the tactical level [15].

Agile Practices Selection. Figure 3 depicts the relationship between the different agile concepts: *value*, *principle*, *agile feature* and *practice*. They are represented at four different levels, the lower levels contributing to the upper ones. Due to lack of space, only practices that help to achieve the two principles contributing to a value are shown. In Fig. 3, the value *"Individual and interaction over processes and tools" is contributed by two principles: "Build project around motivated individual. Give them the environment and support they need and trust them to get the job done"* and *"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation"*. Each principle is emphasized by multiple agile features such as *"Trust"*, *"Support"*, *"Good environment"* and *"Motivated individuals"*. Each agile feature can be achieved by multiple agile practices of Scrum and XP. For example, we can achieve the feature *"Trust"* by using the practices "Collective ownership" and "Sign up".

4.3 Operational Level Application: Towards Practices Implementation

This section presents how our framework helps agile practitioners in adopting agile methods at the operational level. The relationships, particularly dependencies, between roles required to perform the practices will be described first. Then we will explain how the SD model can help practitioners to identify vulnerabilities and what they should do to avoid them.

Dependency Between Roles: How to Define Depender and Dependee? In agile methods, the many roles can be grouped into four main categories:

- Product owner: the role who demands (and pays) for software;
- Coordinator: the role (Scrum Master or XP Coach) who coordinates the different roles to make sure that development is running smoothly and all practices are adopted correctly;
- Development team: the group of people with many roles (developer, tester, designer, etc.) who perform a set of activities (design, develop, test, etc.) to build software;
- Agile team: all roles in the team (Development team and Coordinator) who work together to deliver software answering to the demand of product owner.

There may be multiple dependencies, back and forth, between the roles to perform a practice. However, it is always possible to identify which dependency is crucial and which role plays an important part for a successful practice adoption. For instance, for "Daily meeting", one can easily spot that the participation of the team is important but the most important part is the organization by the Scrum Master. A ceremony which is well-organized and well-guided can go smoothly and members will get involved easily. This is to say that, to successfully adopt "Daily meeting", **Agile team** is the **depender** who depends on **Scrum Master** who is the **dependee**. On the other hand, for the practice "Sign up" or "Pair programming", the guidance from the coordinator is needed but its success depends strongly on the effort and the motivation of the team. In this case, **Scrum Master** or **XP Coach** is the **depender** and **Agile team** is the **dependee**.

In the aforementioned examples, we did not mention Product Owner as the depender since Product Owner is not the main beneficiary of these practices. Product Owner does not care whether practices "Daily meeting" or "Sign up" or "Pair programming" are adopted successfully or not. The practice in Scrum and XP that give benefits to Product Owner is "Frequent release" since it can help to deliver the product frequently to the Product Owner.

Table 4 exposes the dependency of the selected practices discussed in the previous section. A whole list of dependencies between roles to perform Scrum and XP practices and justification can be found at https://goo.gl/nRP6VN (Sect. 9).

Dependency Representation in i*. A role, in the meta-model, is in charge of carrying out practices. The description of all the dependencies, critical roles and artifacts can help the organization and the software team to better analyze the chances of success or the probability of failure and thus be prepared to mitigate the risks. For instance, if many critical practices heavily depend on a particular role, the probability of failure increases when that role is not reliable.

Table 4. Role dependencies.

Depender	Practice	Dependee	Justification
XP coach	Collective ownership	Development team	Depends strongly on team cooperate with each other on the same work
XP coach	Sign up	Development team	Depends strongly on team to select task voluntarily
XP coach	Pair programming	Development team	Depends strongly on team to work together on the same computer
Development team	Sustainable pace	XP coach	Depends strongly on the guidance
Development team	Three question	Scrum master	Depends strongly on the guidance
Development team	Daily meeting	Scrum master/XP coach	Depends strongly on the guidance
Agile team	Backlog grooming	Product owner	Depends strongly on product owner to make sure that product backlog remain correct
Agile team	Planning poker	Product owner	Depends strongly on product owner to explain each user story
Agile team	Backlog	Product owner	Depends strongly on product owner to create the list of user stories and prioritize them
Agile team	User stories	Product owner	Depends strongly on product owner to write user stories

Consequently, software teams must carefully assign it or adopt better practices to reduce or avoid such risks. By visualizing this concept in i*, one can easily understand how each role is involved in a particular process to achieve a goal.

Figure 4 shows the dependencies between roles to perform the practices in Scrum and XP, illustrated in Fig. 3, at the operational level.

When practices in different methods are merged, so will do the roles. The software team has to decide which roles to keep and which to eliminate. In our example, the XP coach depends on the developer in only two practices from XP; i.e. to perform "Pair programming" and "Collective ownership". Since the XP Coach is just a depender which is not crucial, it can be replaced by the Scrum Master. As for the developer role, even if it is necessary to perform the practices, it is already included in the software team.

In the figure, Agile team (Scrum Master and Development team) depends on the Product Owner to perform the "Backlog grooming", "Creating user stories",

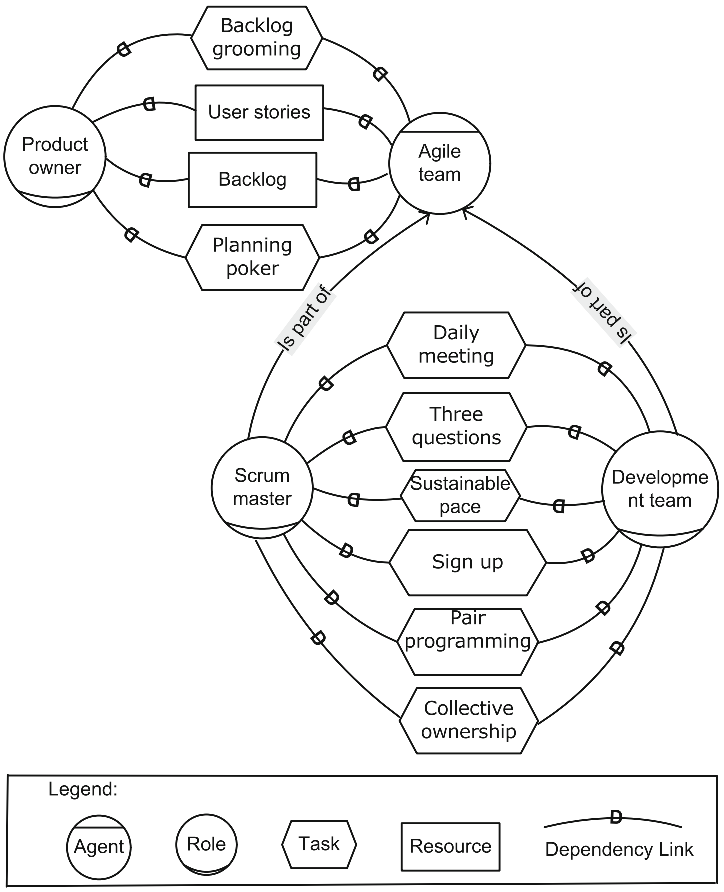


Fig. 4. Scrum methodology at the operational level.

"Building the backlog" and "Planning poker" practices. Scrum Master depends on Development team to do the "Sign up", "Pair programming" and "Collective ownership". Development team depends on Scrum Master to make sure that "Daily meeting", "Three questions" and "Sustainable pace" are used correctly.

Vulnerability Check. In order to achieve all the goals in Fig. 3 using Scrum and XP, all the 10 practices must be adopted successfully. Among them, 6 practices that depend on Scrum Master and Development team do not seem to present any problem. But the four other practices depending on the Product Owner may have vulnerability. In general, the Product Owner is not always available and he/she is limited in time. Adopting four practices at the same time might be difficult for him/her. Based on this vulnerability, it is necessary that the Scrum Master manages to communicate effectively with the Product Owner despite his/her tight schedule.

5 Conclusion

This paper introduced a new framework for socio-intentional based partial agile methods adoption. The framework offers three main contributions: a goal-oriented meta-model, a goal and social representation by using the i* framework and an approach for practices selection and adoption. We based our meta model on the Agile Manifesto concepts and different popular agile methods. It allows the software team to quickly understand not only the objective (goal) of each concept (i.e., value, principle, agile feature and practice, role and artifact), but also their relationships and dependencies. To illustrate such relationships between agile concepts on an intentional and social perspective, we adopted the i* framework. This enables a visualization more effective for the software team to quickly identify the practices to adopt based on their goals. Vulnerabilities can also be easily spotted.

To validate the framework, we took the agile values and principles of the Agile Manifesto as well as the 23 agile features from [46] and the 27 practices from Scrum and XP as instances of our meta-model. Results show that all these instances are objects of the right classes having the right relationships between each other as represented in the meta-model. All these instances can also be visualized using i* and facilitate the agile practices selection and adoption process.

This framework helps agile practitioners to partially adopt methods and fulfill their goal. We acknowledge, however, that an in-depth study on agile methods is needed to improve our meta-model with a systematic review and a qualitative research study. At the time of writing, this systematic literature review is already under progress.

To make sure that the software team can successfully adopt agile methods, we will also add an evaluation of the specific adoption process. We aim not only at offering a roadmap for the software team to adopt agile methods, but also a set of performance indicators during the adoption process.

A wider variety of choices in terms of practices should also be offered. To allow such possibility, one could introduce a weighting scheme, between different alternatives, such as a risk indicator, when adopting a particular practice. Finally, in the long run, we aim at developing a software tool for showcasing our meta-model.

References

1. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley Longman Publishing Co. Inc., Boston (2000)
2. Palmer, S.R., Felsing, M.: A Practical Guide to Feature-Driven Development, 1st edn. Pearson Education, New York City (2001)
3. Stapleton, J.: DSDM: The Method in Practice. Addison-Wesley Longman Publishing Co. Inc., Boston (1997)
4. Cockburn, A.: Surviving Object-oriented Projects: A Manager's Guide. Addison-Wesley Longman Publishing Co. Inc., Boston (1998)

5. Schwaber, K., Beedle, M.: Agile Software Development with Scrum, vol. 1. Prentice Hall, Upper Saddle River (2002)
6. Fowler, M., Highsmith, J.: The agile manifesto. Softw. Dev. **9**, 28–35 (2001)
7. Abrahamsson, P., Warsta, J., Siponen, M.T., Ronkainen, J.: New directions on agile methods: a comparative analysis. In: Clarke, L.A., Dillon, L., Tichy, W.F. (eds.) Proceedings of the 25th International Conference on Software Engineering, 3–10 May 2003, pp. 244–254. IEEE Computer Society, Portland (2003)
8. Tripp, J.F., Armstrong, D.J.: Exploring the relationship between organizational adoption motives and the tailoring of agile methods. In: 47th Hawaii International Conference on System Sciences (HICSS), pp. 4799–4806. IEEE (2014)
9. Campanelli, A.S., Parreiras, F.S.: Agile methods tailoring - a systematic literature review. J. Syst. Softw. **110**, 85–100 (2015)
10. Esfahani, H.C., Yu, E.S.K., Annosi, M.C.: Towards the strategic analysis of agile practices. In: Nurcan, S. (ed.) Proceedings of the CAiSE Forum 2011, Volume 734 of CEUR Workshop Proceedings, London, UK, 22–24 June 2011, pp. 155–162. CEUR-WS.org (2011)
11. Henderson-Sellers, B., Gonzalez-Perez, C.: A comparison of four process metamodels and the creation of a new generic standard. Inf. Softw. Technol. **47**, 49–65 (2005)
12. Mikulėnas, G., Butleris, R., Nemuraitė, L.: An approach for the metamodel of the framework for a partial agile method adaptation. Inf. Technol. Control **40**, 71–82 (2011)
13. Esfahani, H.C., Cabot, J., Yu, E.S.K.: Adopting agile methods: can goal-oriented social modeling help? In: Loucopoulos, P., Cavarero, J. (eds.) Proceedings of the Fourth IEEE International Conference on Research Challenges in Information Science, RCIS 2010, Nice, France, 19–21 May 2010, pp. 223–234. IEEE (2010)
14. Madi, T., Dahalin, Z., Baharom, F.: Content analysis on agile values: a perception from software practitioners. In: 2011 5th Malaysian Conference on Software Engineering (MySEC), pp. 423–428. IEEE (2011)
15. Kiv, S., Heng, S., Kolp, M., Wautelet, Y.: An intentional perspective on partial agile adoption. In: Proceedings of the 12th International Conference on Software Technologies - Volume 1, pp. 116–127. ICSOFT, INSTICC, SciTePress (2017)
16. Yu, E.S.: Social modeling and *i**. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 99–121. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02463-4_7
17. Jacobson, I., Ng, P.W., Spence, I.: The essential unified process-a fresh start for processd. Dr. Dobbs J. **31**, 40+ (2006)
18. Ambler, S.: The Agile Unified Process (AUP). Ambysoft (2005). http://www.agilealliance.hu/materials/books/SWA-AUP.pdf
19. Kroll, P., MacIsaac, B.: Agility and Discipline Made Easy: Practices from OpenUP and RUP (Addison-Wesley Object Technology (Paperback)). Addison-Wesley Professional, Boston (2006)
20. Ambler, S.: Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. Wiley, Hoboken (2002)
21. Ahmad, M.O., Markkula, J., Oivo, M.: Kanban in software development: a systematic literature review. In: 2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 9–16. IEEE (2013)
22. Liker, J.K.: The Toyota Way. Esensi (2004)
23. Poppendieck, M., Poppendieck, T.: Lean Software Development: An Agile Toolkit. Addison-Wesley, Boston (2003)

24. Chiarini, A.: Lean Organization: From the Tools of the Toyota Production System to Lean Office. Springer, Heidelberg (2013). https://doi.org/10.1007/978-88-470-2510-3
25. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press (2004)
26. Henderson-Sellers, B., Ralyté, J., Ågerfalk, P.J., Rossi, M.: Situational Method Engineering. Springer, Heidelberg (2014)
27. Lin, J., Yu, H., Shen, Z., Miao, C.: Using goal net to model user stories in agile software development. In: 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2014, Las Vegas, NV, USA, 30 June–2 July 2014, pp. 1–6. IEEE Computer Society (2014)
28. Shen, Z., Miao, C., Tao, X., Gay, R.: Goal oriented modeling for intelligent software agents. In: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2004), pp. 540–543. IEEE (2004)
29. Bézivin, J.: In search of a basic principle for model driven engineering. Novatica J. Special Issue **5**, 21–24 (2004)
30. Schuppenies, R., Steinhauer, S.: Software process engineering metamodel. OMG group, November 2002
31. Damiani, E., Colombo, A., Frati, F., Bellettini, C.: A metamodel for modeling and measuring scrum development process. In: Concas, G., Damiani, E., Scotto, M., Succi, G. (eds.) XP 2007. LNCS, vol. 4536, pp. 74–83. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73101-6_11
32. Wautelet, Y., Heng, S., Kiv, S., Kolp, M.: User-story driven development of multi-agent systems: a process fragment for agile methods. Comput. Lang. Syst. Struct. **50**, 159–176 (2017)
33. Pourmasoumi, A., Kahani, M., Bagheri, E., Asadi, M.: Process fragmentation: an ontological perspective. In: Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., Ma, Q. (eds.) CAISE 2015. LNBIP, vol. 214, pp. 184–199. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19237-6_12
34. Wautelet, Y., Heng, S., Kolp, M., Mirbel, I.: Unifying and extending user story models. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 211–225. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07881-6_15
35. Wautelet, Y., Heng, S., Kolp, M., Mirbel, I., Poelmans, S.: Building a rationale diagram for evaluating user story sets. In: Tenth IEEE International Conference on Research Challenges in Information Science, RCIS 2016, Grenoble, France, 1–3 June 2016, pp. 1–12. IEEE (2016)
36. Wautelet, Y., Heng, S., Hintea, D., Kolp, M., Poelmans, S.: Bridging user story sets with the use case model. In: Link, S., Trujillo, J.C. (eds.) ER 2016. LNCS, vol. 9975, pp. 127–138. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47717-6_11
37. Seidita, V., Cossentino, M., Chella, A.: A proposal of process fragment definition and documentation. In: Cossentino, M., Kaisers, M., Tuyls, K., Weiss, G. (eds.) EUMAS 2011. LNCS (LNAI), vol. 7541, pp. 221–237. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34799-3_15
38. Yu, E., Mylopoulos, J.: Understanding "why" in software process modelling, analysis, and design. In: Proceedings of the 16th International Conference on Software Engineering, pp. 159–168. IEEE Computer Society Press (1994)
39. OMG: Software & systems process engineering meta-model specification. Version 2.0. Technical report, Object Management Group (2008)

40. Faulkner, S., Kolp, M., Wautelet, Y., Achbany, Y.: A formal description language for multi-agent architectures. In: Kolp, M., Henderson-Sellers, B., Mouratidis, H., Garcia, A., Ghose, A.K., Bresciani, P. (eds.) AOIS-2006. LNCS (LNAI), vol. 4898, pp. 143–163. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77990-2_9
41. ISO/IEC: ISO/IEC 12207:2008: Systems and software engineering - software life cycle processes (2008)
42. Van Loon, H.: Process Assessment and ISO/IEC 15504: A Reference Book. Springer, Heidelberg (2004)
43. Madeyski, L.: Test-Driven Development: An Empirical Evaluation of Agile Practice, 1st edn. Springer Publishing Company, Incorporated, Heidelberg (2010)
44. Sidky, A.S., Arthur, J.D., Bohner, S.A.: A disciplined approach to adopting agile practices: the agile adoption framework. ISSE **3**, 203–216 (2007)
45. Karlström, D., Runeson, P.: Integrating agile software development into stage-gate managed product development. Empirical Softw. Eng. **11**, 203–225 (2006)
46. Laanti, M., Similä, J., Abrahamsson, P.: Definitions of agile software development and agility. In: McCaffery, F., O'Connor, R.V., Messnarz, R. (eds.) EuroSPI 2013. CCIS, vol. 364, pp. 247–258. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39179-8_22
47. Pressman, R.S.: Software Engineering: A Practitioner's Approach. Palgrave Macmillan, Basingstoke (2005)
48. Cohn, M.: User Stories Applied: For Agile Software Development. Addison Wesley Longman Publishing Co. Inc., Redwood City (2004)
49. Yu, E., Giorgini, P., Maiden, N., Mylopoulos, J.: Social Modeling for Requirements Engineering. MIT Press, Cambridge (2011)
50. AgileAlliance: Subway map to agile practices (2005). https://www.agilealliance.org/
51. VersionOne: 10th annual state of agile development survey (2016)
52. Schwaber, K., Sutherland, J.: The scrum guide. Scrum Alliance 21 (2011)

Using Semantic Web to Establish Traceability Links Between Heterogeneous Artifacts

Nasser Mustafa[1](✉) and Yvan Labiche[2](✉)

[1] University of Nottingham, 199 Taikang East Road, Ningbo, China
Nasser.mustafa@nottingham.edu.cn
[2] Carleton University, 1125 Colonel by Dr, Ottawa, Canada
labiche@sce.carleton.ca

Abstract. Semantic Web enables the users of the World Wide Web (WWW) to create non-traditional data repositories. The data can be linked in a flat hierarchy structure that allows the extensibility of data without the need for changing the structure itself. The linked data along with other rules can be used to infer or extract other data. We propose a semantic web technique that employs the Resource Description Framework (RDF) for building a trace links taxonomy. The taxonomy can be utilized to link heterogeneous artifacts coming from different domains of expertise. This technique allows users to refer to any trace link type in the taxonomy using a unique Uniform Resource Identifier (URI). The taxonomy can also be integrated to a traceability framework using the Open Service for Lifecycle Collaboration (OSLC) in order to accommodate the traceability of heterogeneous artifacts. We present validation criteria for validating the taxonomy requirements and validate the solution through a set of test cases. A simple case study is used in order to provide meaningful results.

Keywords: Traceability · Trace links · Semantic web · Taxonomy
Linked data · Resource description factor
Open Service for Lifecycle Collaboration

1 Introduction

Capturing traceability information among artifacts ensures product quality and assists tracking functional and non-functional requirements, and performing system validation and impact analysis. Software artifacts can be produced during Requirement Engineering (RE), Model Driven Engineering (MDE), and Systems Engineering (SE). They are heterogeneous in nature since they are produced by different tools, and for different system disciplines. Establishing relationships between these artifacts requires different types of trace links with precise semantics. Unfortunately, there is a lack of consensus among software practitioners for defining precise trace links semantics. This is an issue since using different, either overlapping or conflicting semantics for trace links can have adverse effect on product quality [2]. The heterogeneity of artifacts that are involved in the development of a complex system requires various types of trace links. The variations between RE, MDE, and SE domains require different types of trace links

E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 91–113, 2018.
https://doi.org/10.1007/978-3-319-93641-3_5

to relate their artifacts. There are situations in which ambiguity exists in capturing traceability information among artifacts as a result of the absence of a reference model that describes the various types of trace links and their exact purposes.

This research paper is motivated by the need for a traceability framework that can be used to accommodate traceability of heterogeneous artifacts. In order to demonstrate this need, we consider the example of a full flight simulator. A full flight simulator typically includes software (e.g., simulating specific hardware components, simulating missions), visuals and audio (e.g., audio rendering of sound inside the flight deck, video rendering of typical airports), mechanical systems (e.g., to provide accurate force feedback to the pilot, to provide motion for the entire flight deck simulator), communication systems (e.g., air traffic). In this example, several heterogeneous models need to be related to one another during system engineering. For instance, a model would be for mission simulation that can be used to create specific training scenarios. One model would be a Simulink hydraulic actuation system; one model would be a simulation model of a hardware component; one (graphical) model would be used to represent takeoff and landing characteristics (e.g., visuals, air traffic data) of typical airport; one model would record the requirements for the whole system and the requirements for its many (software and hardware) parts; one model would record data about verification and validation activities; one model would record faults and failures. These models are also heterogeneous because they are not typically specified using a common notation and semantic, i.e., a common metamodel. Indeed, some software elements can be specified with the UML [12], some system levels characteristics can be modeled with SysML [13], some hardware elements can be modeled with Simulink models, and some information can even be provided in plain language (e.g., requirements). In this example, different trace links are needed based on users' needs. For instance, a user might need to trace the effect of changing the lever position in the pilot cockpit to the wing's slats. Since this process involves the generation of several signals within the simulator, therefore, many types of trace links are required based on the granularity level of traceability. At higher level of traceability, a user might be interested in identifying only the components (models) involved, and at a fine-grained level, the user might be interested in determining the involved activities that are realized be an UML activity diagram. Those two levels require different types of trace links with different semantics. This scenario and others encouraged us to build a trace links taxonomy that combines all trace links across different domains.

The rest of this paper is structured as follows. Section 2 presents related work on trace links and their limitations. Section 3 highlights the requirements for trace links taxonomy and introduces the RDF technique. Section 4 explains the design decisions for the proposed taxonomy. Section 5 describes the design implementation, focusing on the benefit of using RDF in building the trace links taxonomy. Section 6 shows our validation criteria, the requirements validation, and discusses the validation of the taxonomy through a case study. Section 7 concludes the paper.

2 Literature Review

We conducted a systematic literature review [17], following standard and well-recognized principles for conducting systematic literature reviews [19], on the topic of traceability. The review covered papers published between the years 2000–2016 in five major computing libraries (i.e., IEEE Xplore, ACM, Google Scholar, Science Direct, and Springer). We specified the following search string in order to extract the traceability publications in RE, MDE, and SE: Traceability AND (Heterogeneous OR Modeling OR Models OR MDE OR Model Driven OR Trace Link OR Requirement Engineering OR Systems Engineering OR Software Engineering). Based on our review, we identified some research papers that define traceability and traceability relations [2, 3, 5, 7, 21–27], other papers that classify or identify some types of trace links [2–11, 14–16, 18, 23, 24, 29–32], and some papers that discuss the need for trace link semantics [20, 28, 32–34]. Although these papers provide valuable information on traceability definitions and classifications, we couldn't find any paper that suggests a technique for building a trace links taxonomy that combines the trace links from all domains into a unified taxonomy. Most of these studies are confined to defining trace links and their semantics only for a specific problem or domain, i.e., solutions are problem or domain specific. For instance, there is a great deal of effort on classifying traceability links and their usage in RE [2, 7], though classifications only apply to RE.

This section elucidates important aspects about our review of traceability in RE, MDE, and SE. Section 2.1 discusses traceability and trace links definitions while Sect. 2.2 discusses existing trace links classifications in RE, MDE, and SE. The review in this section will be used as a core for our work in order to collect all trace links types for building the taxonomy.

2.1 Traceability Definitions

Traceability is defined by the IEEE [22] as "the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another". This definition applies to traceability in RE, MDE, and SE as well. The IEEE definition is extended to include other types and subtypes of relationships. Cleland-Huang and colleagues [23] describe trace link semantics and types. A trace link semantics refers to the purpose or meaning of the relationship between associated artifacts. A trace link type refers to the characterization of all trace links that have a similar structure (syntax) and/or purpose (semantics). The description of a trace link type encapsulates the definition of a trace link semantics since it is explained based on the link's semantic role, and may include other properties such as the rationale for creating a trace link. For instance, all trace links that relate two artifacts where one artifact is derived from another have the trace link type "derived from". The "derived from" type represents the meaning of the relation between such artifacts. Therefore, we might have similar or extended types of trace links among the RE, MDE, and SE domains. Readers should note that we use a notion of trace link and relation interchangeably, however, there is a difference between both terms since the latter refers to all trace links created between two sets of trace artifact types [23].

In RE, several types of trace links are introduced as a result of traceability definitions. Gotel and colleagues [3] defined traceability as the ability to describe and follow the life of a requirement in both forward and backward directions. In this context, a pre-requirement specification refers to the aspects of a requirement's life prior to its inclusion in the requirement specification, and a post-requirement specification refers to the aspects of a requirement's life that result from its inclusion in the requirement specification [24]. Also, there are the notions of vertical and horizontal traceability [7, 23, 25, 37]. Horizontal traceability refers to tracing artifacts created in the same system lifecycle phase, or at the same level of abstraction. For instance, tracing two requirements based on the "derived from" relationship is horizontal. Vertical traceability refers to tracing artifacts created in different phases or at different levels of abstraction, such as tracing a requirement in the requirement specification phase to a test case in the testing phase.

In MDE, Aizenbud-Reshef and colleagues [26] defined traceability as "any relationship that exists between artifacts involved in the software-engineering life cycle". This definition is broader than the RE definition since it assumes other types of trace links such as explicit links which can be generated during model transformation, implicit links which are computed based on existing information, and statistical links that can be inferred based on history.

In SE, Mason [5] extended the notions of vertical and horizontal traceability by introducing the terms Micro, Macro, Inter, and Intra traceability. The Micro and Macro terms are introduced to differentiate traceability within and across decomposition levels. The Intra and Inter terms are introduced to differentiate traceability within and across system descriptions (i.e., interactions between systems). For instance, the Inter-Micro-Horizontal traceability refers to the ability to describe and navigate relationships across system descriptions, within a decomposition level, between development or assessment artifacts of the same type.

Our aim is to build a trace links taxonomy which has well-defined semantics and that encompasses various types of trace links in the RE, MDE, and SE disciplines. This is important for many reasons. First, in RE, many artifacts are produced during requirements elicitation, analysis, and validation, hence, require different types of trace links with different semantics. Second, in MDE, which permits model transformations, a large number of trace links is required to link artifacts in source and target models, some of which are generated automatically while others require manual generation; relating these artifacts requires well-defined semantics for trace links, which are slightly different from what one can define in RE or SE. Third, in SE, the development of a complex system involves the generation of heterogeneous artifacts as a result of using different modeling tools for modeling different aspects of the system, from different disciplines (e.g., electrical, software). Fourth, comprehending the rationale for creating different types of trace links among artifacts at different levels of granularity requires well-defined trace links semantics. Fifth, there are situations that require many types of trace links in the same domain but for different purposes. For instance, when linking two requirements, a requirement derived from another requires a different trace link than a requirement clarified by another. Sixth, the meaning of a trace link can be viewed differently by different stakeholders. For instance, a trace link between a requirement and a design element may be viewed by a designer as a constraint the requirement imposes on the design element, while an end user might view the same link as a design element produced by the requirement [2]. Finally, with the various types of modeling tools across different domains, it is a necessity to have a trace links taxonomy that can be integrated with other API's. In other words, we need a portable taxonomy that can be integrated easily with other tools.

The contribution of this paper includes the followings. First, we propose requirements for a trace links taxonomy. Second, we offer a technique to build a trace links taxonomy which has well-defined semantics and that can accommodate the classification of trace links in RE, MDE, and SE. The taxonomy employs the Open Service for Lifecycle Collaboration (OSLC), and the Resource Description Framework (RDF) [38] for defining a set of properties and their values for each trace link. Third, we validate the taxonomy through a case study that requires heterogeneous artifacts from multiple domains.

In an effort to have more insight about trace links and their classifications we conducted a systematic literature review about traceability aspects [17]. Based on our review, we identified some research papers that define traceability and traceability relations, papers that classify or identify some types of trace links, and papers that discuss the need for trace links semantics. Although these papers provide valuable information on traceability definitions and classifications, we couldn't find any paper that suggests a technique for building a trace links taxonomy that combines trace links from all domains. Most of these studies are confined to defining trace links and their semantics only for a specific problem or domain, i.e., solutions are problem or domain specific. For instance, there is a great deal of effort on classifying traceability links and their usage in RE [2, 7], though classifications only apply to RE.

2.2 Traceability Classifications

In general, relations between artifacts are classified based on the development phase or the abstraction level (i.e., horizontal and vertical). However, other classifications are introduced to fit the RE, MDE, and SE needs. The trace links classifications which we found are either problem oriented, i.e., tailored to special cases, and are not applicable within a general context (e.g., between requirement and source code), or target one domain only (e.g., RE or MDE). This section summarizes our effort in collecting and organizing these classifications in order to build a trace links taxonomy. We first discuss each domain separately (RE, MDE and SE in this order) and then summarize our observations [1].

Requirement Engineering Classifications. In RE, relationship between artifacts are discussed extensively [2–4, 6–11, 14–16, 18, 23, 24, 29, 39]; among these papers, we found two papers that discuss trace links classifications [2, 7]. Spanoudakis and Zisman [7] classified requirement traceability links into eight categories, which include various link types, based on their support to certain software activities such as analysis, validation, or supporting stakeholders decisions. These links include the following types and the information is summarized in Tables 1 and 2:

- The dependency links which relate artifacts in which the existence of one artifact relies on the existence of the other. This type can be used to relate requirements to each other, or requirements and design elements (artifacts) such as decision objects. Dependency relations are one of the most widely used dependency links in RE and have different uses and forms [7]. For instance, Xu and Ramesh [8] use dependency relations in workflow management systems between business process objects, decision objects, and workflow system objects. Dependency relations are used in product and service families [29] to support the management of variability, i.e., ensuring that the changed artifacts reflect the intended system functionality. Knethen and colleagues [35] suggested their use between documentation entities such as requirements and use cases, and logical entities such as functions for fine grained impact analysis. Pohl and Alexander [9, 10] use them to link requirement scenarios and code, and Riebisch and Philippow [11] use them to support the design and implementation of product lines. Other forms of dependency links are suggested in the literature. For instance, Spanoudakis and colleagues [6] refer to them as requires-feature-in relations, as they link parts of use case specifications to customer requirements specifications. Also, they are called causal conformance by Maletic et al. [14] who use them to link documents that represent an implied ordering (e.g., bug reports cannot be produced before implementation report). Gotel and colleagues [16] referred to them as developmental relations which are used to trace requirements to other artifacts in another phase of the development lifecycle. Finally, they are referred by Constantopoulos et al. [18] to as correspondence relations which link requirements, design, and code artifacts.

Table 1. Trace links classifications in RE, source [1].

	Trace link type							
[2]	Product-related			Process-related				
	Evolution	Rationale	Dependency	Satisfaction				
	Derive, elaborate, depend-on	Select, affect	Is-a, part-of, contain, used-by, performed-by	Define, allocate-to, depend-on, created-by, verify, generate				
[7]	Dependency	Evolution	Generalize/refine	Satisfaction	Overlap	Conflict	Rationale	Contribution
		Replace, based-on, formalize, elaborate				Based-on, affect, resolve, generate		

- The evolutionary relations are used to link requirements in which one requirement replaces another. This category contains the replace, based-on, formalize, and elaborate trace links. Pinheiro [15] showed the use of replace and abandon trace links during requirements evolution. A requirement is replaced by another if a mistake is discovered the original requirement will be abandoned. Gotel [16] called the evolution relations temporal relations which refer to linking requirements in terms of their historical order. Maleic and colleagues [14] called the evolution relations non-causal conformance relations to link documents which conform to each other.

- The generalization/refinement relations show how complex system components can be divided into other artifacts, or how one artifact can be refined by another. In Ramesh and Jarke's classification [2], generalization/refinement is considered a dependency abstraction link. Gotel [16] refers to them as containment relations since they are used to link composite artifacts and their components.
- The satisfiability relations link artifacts that are constrained by each other, e.g., a requirement that complies with the conditions of another requirement. This type is classified as a product-related trace link to relate requirements to design artifacts [2]. Satisfiability has sub-types such as the establish (cardinality 1–1 between two artifacts) and contribute (cardinality 1-m between artifacts) relations [28]. Pinheiro [15] defined satisfiability based on derivation, e.g., if a requirement is satisfied then its derivation is satisfied, and refinement, i.e., if a requirement refines another requirement, then satisfying the first requirement, implies satisfying the second. between artifacts of common features (e.g., linking a goal specification in an i* model and a use case in a UML model) [36]. Spanoudakis et al. [6] use the overlap relations in an analysis model between use cases and classes. Gotel and Finkelstein [16] called them adopts relations; they are used between artifacts in which a target artifact embeds information of the source artifact.
- The conflict relations link two artifacts that have a conflict, such as two requirements that are conflicting with each other [2]. Special types of conflict relations such as based-on, affect, resolve, and generate are used to facilitate conflict

Table 2. Other classifications for RE trace links, source [1].

	Trace link type					
[3]						Contribution
[6]	Requires-feature-in				Overlap	
[8]	Dependency				Rationale	
[9]	X					
[10]	X					
[11]	X					
[14]	Causal-dependency conformance	Non-causal conformance				
[15]				Satisfy Derive Refine		
[16]	Developmental	Temporal	Containment		Adopt	
[18]	Correspondence					
[20]					X	
[28]				Satisfy Establish Contribute		
[35]						Inconsistency
[36]					X	

resolution between conflicting artifacts. Kozlenkov and Zisman [39] referred to conflict relations as inconsistency relations. For instance, inconsistency relations are established when two similar goals in a specification or different specifications cannot be achieved.

- The rationalization relations link two artifacts in which one of them captures the rationale behind the creation or evolution of the other. Letelier [20] used this type to relate rationale specification artifacts (e.g., decisions, assumptions) to software specifications at different levels of granularity (e.g., document or part of a document, diagram, or a model). Rationalization relations are used also to relate design rationales to design artifacts [8].
- The Contribution relations relate requirements and their stakeholders [3], for instance to link requirements to the stakeholders who contributed them.

Another classification for trace links in RE is introduced by Ramesh and Jarke [2]. Their classification is based on a study about the use of trace links by different organizations that involve high-end and low-end users with respect to their traceability practices. They classified traceability links into two main categories: process-related and product-related links. The process-related links can be discovered by observing the history of operations performed in a process. The product-related links describe the relationships between artifacts independent of their creation. Furthermore, the authors identified sub-categories of these two main categories. The process-related category is divided further into evolution links and rationale links, which we described earlier. On the other hand, the product-related links are decomposed into two main types: satisfaction links and dependency links, which we described earlier. The authors deduced other types of relations from the abovementioned categories based on the needs of low-end and high-end users. For instance, with respect to low-end users', the original or derived requirements can be allocated-to system components that interface-with external systems. Also, requirements can be developed-for compliance-verification-procedures (e.g., test, simulation, and prototype). Compliance-verification-procedures generate change proposals or used-by resources. With respect to high-end users, traceable artifacts (e.g., requirements, components, designs) are based-on a rationale. Decisions depends-on assumptions, or select or evaluate alternatives. Also, decisions may affect requirements, and arguments oppose or support alternatives.

Model Driven Engineering Classification. In MDE, artifacts are modeled using different modeling languages such as the UML [40, 41]. During model transformation, trace links between these artifacts are generated explicitly by adding additional code into the transformation, or implicitly through the transformation tool [31]. Paige and colleagues [4] classified MDE trace links into implicit and explicit trace links. Implicit trace links are classified based on query, transformation, composition (merging), update, deletion and creation, model-to-text, and sequences operations. Explicit trace links are classified as model-to-model links which relate MDE artifacts with each other, and model-to-artifact links which relate MDE artifacts with non-MDE artifacts such as linking a UML model to its requirement(s). The model-to-model links are further classified into static and dynamic links. A static link represents a relationship that stays the same over time between models elements such as consistent-with (e.g., two models remain consistent with each other), and dependency in which the structure and meaning

Table 3. MDE trace links classifications, source [1].

	Trace link type				
[4]	Implicit	Explicit			
		Model-to-model			Model-to-artifact
		Static		Dynamic	Satisfy, allocated-to, explain, perform, support
		Consistent-with	Dependency	Call, notify, generate	
			Export, usage, is-a, has-a, part-of, import, refine		

of one model depend on another model. This type is further classified into the following trace links: is-a (sub-typing), has-a (e.g., references), part-of, import, export, usage, and refinement. A dynamic link represents a relationship that might evolve over time. This category has several types of links such as calls (e.g., a model calls the behaviors provided by another), notifies (e.g., changed artifacts that need intervention), and generates (e.g., links two models where one model produces the other). The model-to-artifact category contains the satisfies trace link which indicates that an artifact such as a requirement is satisfied by a model, allocated-to which relates information in a non-model artifact to a model that represents that information, performs which relates a task to a model that carries the task, explains and supports trace links which are used when a model is explained by a non-model artifact. Table 3 depicts the existing classifications of MDE trace links.

In SysML [41], the classification of trace links between model elements follows the same RE pattern. Requirements in SysML are classified into different categories using a Stereotype. For instance, requirements can be stereotyped to represent operational, functional, performance, design constraints, reliability, and maintainability requirements. The SysML requirement diagram models these requirements in hierarchies. Therefore, the semantic of a trace links in SysML depends on the type of the relationship between two requirements, or between a requirement and a design element or a test case for example. There are four main types of trace links in SysML: A derived trace link links a requirement to its derived one; A satisfy trace link relates a requirement to a design element that satisfies it such as a Use case; A verify trace link links a requirement to a test case that verifies it; A refine trace link links a requirement to a design element that refines it. It is important to differentiate between the usage or the semantic of derive and refine trace links. A derive trace link is used to link two requirements while a refine trace link links a requirement to a design element that refines it or vice versa such as a text requirement with an activity diagram [42].

Systems Engineering Classifications. In SE, Mason et al. [5] define directional and temporal traceability. They also extended the definitions of vertical and horizontal traceability by introducing the terms micro, macro, inter, and intra. The micro and macro terms differentiate traceability within and across decomposition levels. Intra and inter differentiate traceability within and across system descriptions (i.e., interactions between systems). For instance, the inter-micro-horizontal traceability refers to the ability to describe and navigate relationships across system descriptions, within a decomposition level, between development or assessment artifacts of the same type. This is illustrated in Table 4. Temporal traceability represents the links between synchronized artifacts, such as linking an artifact and its subsequently revised one, in a model based on a time event. Many trace link types are stemed from the formentioned terms as depectited by Table 4.

Table 4. Trace links types in systems engineering, source [1].

	Trace link type				
[5]	Temporal	Directional			
		Vertical		Horizontal	
		Micro	Macro	Micro	Macro
		Inter/intra	Inter/intra	inter/intra	Inter/intra

Final Observations. As evidenced by the above discussions, existing classifications of trace links have the following drawbacks:

- Each classification is either problem specific or domain specific.
- Classifications are inconsistent with respect to their interpretations of link semantics. They often refer to the same semantics with different names. We conjecture this is a side effect of the first drawback. For instance, Spanoudakis and Zisman [7] classified is-a as an evolution link while Paige and colleagues [4] classify it as dependency link.
- Classifications are redundant, which we conjecture is also a side effect of the first drawback. For instance, the rationale trace links appear in RE, MDE, and SE classifications.
- Classifications don't integrate all usages of a trace link across different domains. In other words, the purpose of a certain trace link does not necessarily appear in all classifications to be used in all domains. To obtain a complete picture of traceability information across those domains requires that we merge (union) classifications (while removing redundancies and inconsistencies).
- There is no tool support for these classifications that would allow a user to navigate or to query about certain links across different domains.

3 Taxonomy Requirements

The abovementioned limitations encouraged us to think about a new method for integrating all trace links classifications into a taxonomy that would provide the relationship between different trace links. In the light of the previous discussion, we identified that the new taxonomy shall have the following characteristics, or taxonomy requirements (TRQ):

- TRQ 1: the taxonomy shall provide semantic specifications for trace links that relate various artifact types in different domains and at different levels of granularity.
- TRQ 2: the taxonomy shall address the need for different types of users (e.g., analysts, designers, programmers, testers), and therefore different domains.
- TRQ 3: the taxonomy shall allow the specification of a trace link only once and relate it to different domains without duplications.
- TRQ 4: the taxonomy shall be flexible to allow users to add new properties of trace links without changing the existing structure of the taxonomy.
- TRQ 5: the taxonomy shall be portable enough to allow easy access to local users (i.e., connected to a private network) or global users (i.e., connected to the Internet) because this will facilitate tool integration.
- TRQ 6: the taxonomy shall have a universal format that is not tailored to a specific environment or application.

4 Taxonomy Design

There are two essential components that should exist in order to build our trace links taxonomy: (a) providing a set of controlled vocabulary (Metadata), the controlled vocabulary is a collection of terms that have well-defined descriptions across contexts, and (b) identifying the relationships between these terms, which constitute the taxonomy. A taxonomy or Ontology in a broader context is the knowledge domain which is represented by the collection of terms and the relationships between those terms. An Ontology can be defined using the Web Ontology Language (OWL) [37], which is an extension of RDF. Many organizations standardized their controlled vocabulary and made it available freely for use on the internet. For instance, the Friend of a Friend (FOAF) [43] has standard vocabulary/Ontology for social networks across the web, and the Description of a Project (DOAP) [44] describes open source software projects.

4.1 Design Decisions

Our design method relies on our systematic literature review during which we collected all the terms that refer to trace links types in the SE, MDE and RE domains. We then processed that information as follows:

- Identify all articles that discuss trace links classification in RE, MDE, and SE.
- Identify the terms that describe general types of trace links. Usually, these are nouns or adjectives that describe the relationship between artifacts such as dependency, evolution, and vertical.
- Identify all terms that describe the relationship between specific types of artifacts. These relationships are identified by the role name of the association between artifacts. They are usually represented as verbs such as perform, generate, and depend-on.
- Consider the terms that represent general types as classes, and the terms that represent relationships between specific artifacts as instances. In other words, the term or the relationship that cannot be decomposed into other terms is considered as an

instance. For example, in Table 1, evolution is considered as a class because it is a general type that encompasses other relationships such as derive, elaborate, and depend-on, while derive is an instance since it doesn't includes any other relationship.

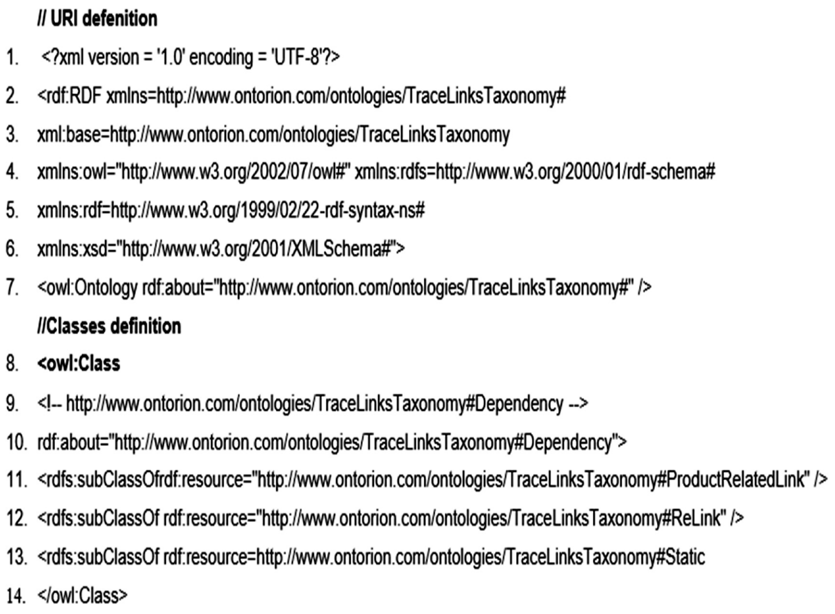
- Provide a naming convention for the general types and the relationships. We have done that by screening all the conjugations that refer to the same type or a relation and give it a unique name. For instance, we considered the evolution and evolutionary terms as identical terms that refer to a general type (i.e., class), which we chose to call evolution. Moreover, we considered the terms perform, performs, and performed as identical terms that refer to a specific relationship (i.e., instance) and we called it perform.
- We removed redundant trace links or relations information that share a purpose or usage. This is achieved by finding all trace links or relations that have identical usage or purpose and then creating one node that specifies them. For instance, in the dependency type that exists in the RE classification [2, 7] and the MDE classification [4], we created a shared node which represents the dependency relation and showed its hierarchical classification with respect to each domain.
- We also provide a set of properties for *instances*. Each *instance* must have unique values that differentiate it from other instances. Even though we limit the properties to include an *Id*, a *name*, a *usage*, a *type*, and a *definition*, other properties can be added. The *name* represents the *instance name*, and the *type* represents the *Class name*.

5 Taxonomy Implementation

The taxonomy utilizes the OSLC and the RDF [38] for relating all trace links. This idea is borrowed from the semantic web technology in which arbitrary data are linked in a flat structure using the RDF, and referenced by the Uniform Resource Identifier (URI). A URI can reference any resource or element such as documents, images, services, a UML diagram, or a group of other resources by assigning it a unique reference.

The RDF has three components: the subject (resource), the predicate (property), and the object (property value). The subject is the element that needs to be described with an assigned unique identifier, the predicate represents the characteristic or feature of that element, and an object is the value of that feature. The object in turns can be a subject that has other properties, which form nested subjects. RDF files are written using the RDF/XML format which is a common format on the web.

RDF files are written using the RDF/XML format or the Web Ontology Language (OWL) which are common formats for writing ontologies. For instance, the code for describing the trace link taxonomy in RDF can be written as shown in Fig. 1. The figure has two parts. The top part (lines 1–8) defines the taxonomy (i.e., schema, syntax, data types). In this part, lines 2 and 8 indicate the namespace of the taxonomy, which represents its URI written in RDF and OWL. The reason for defining another URI in OWL can be seen clearly in the bottom part (lines 9–14). OWL is used to validate the semantics of anything written in RDF. In other words, RDF can be used to describe anything, however, to ensure correct relationships between things written in RDF, OWL must be used to specify these relationships. Therefore, the *owl* tags in lines 10 and 14 imply that a relationship exists between the *dependency* trace link and the other trace links, i.e., *product-related-link, re-link,* and *static* that are enclosed within the owl tags.



```
    // URI defenition
1.  <?xml version = '1.0' encoding = 'UTF-8'?>
2.  <rdf:RDF xmlns=http://www.ontorion.com/ontologies/TraceLinksTaxonomy#
3.  xml:base=http://www.ontorion.com/ontologies/TraceLinksTaxonomy
4.  xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
5.  xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
6.  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
7.  <owl:Ontology rdf:about="http://www.ontorion.com/ontologies/TraceLinksTaxonomy#" />
    //Classes definition
8.  <owl:Class
9.  <!-- http://www.ontorion.com/ontologies/TraceLinksTaxonomy#Dependency -->
10. rdf:about="http://www.ontorion.com/ontologies/TraceLinksTaxonomy#Dependency">
11. <rdfs:subClassOfrdf:resource="http://www.ontorion.com/ontologies/TraceLinksTaxonomy#ProductRelatedLink" />
12. <rdfs:subClassOf rdf:resource="http://www.ontorion.com/ontologies/TraceLinksTaxonomy#ReLink" />
13. <rdfs:subClassOf rdf:resource=http://www.ontorion.com/ontologies/TraceLinksTaxonomy#Static
14. </owl:Class>
```

Fig. 1. RDF example, source [1].

The URI of any trace link in the taxonomy is the concatenation of the taxonomy URI and the trace link name. For instance, the URI of the *dependency* trace link is http://www.ontorion.com/ontologies/TraceLinksTaxonomy#Dependency. The benefit of using the URI is its uniqueness since this URI is a unique identifier for the *dependency* trace link within this taxonomy. In addition, it can be combined with other URI's anywhere to reason or query about the trace link across many namespaces.

The rationale for employing RDF in creating a trace links taxonomy is manifold:

- The RDF eliminates trace links redundancy among different domains, which means resources can be described only once and referenced as many times as we need.
- The RDF supports multiple inheritance in situations where a trace link is classified under more than one category.
- The RDF data is portable, it can be transformed into many formats such as XML, HTML, and OWL.
- The RDF data can be visualized graphically as a directed graph, an undirected graph, or a tree.

- Using the RDF, the taxonomy can be built by referencing trace links from local repositories or external resources such as the Internet.
- The RDF provides the reusability of the same data by different users, which adheres to the principle of open linked data.
- The use of the non-hierarchical RDF structure can provide an easy navigation; a user can reference any trace link in the hierarchy without having any knowledge about its parent(s) or siblings.
- Using the RDF is a step toward standardization and providing semantics for trace links in SE, MDE and RE.
- The RDF data provides simplicity of access since it is machine-readable data that can be shared with others.
- Using the RDF, it is easy to reason (e.g., what, who) about any trace link in the taxonomy.
- Using the RDF, it is easy to query a taxonomy using query services such as SPARQL [38]; the query can be customized based on a user's needs.

We used the Fluent editor application [45] for coding the rules of the taxonomy as described in Sect. 4.1. The editor provides features for authoring complex ontologies that use controlled English as a language for knowledge modeling. It allows users to import and export the knowledge model into different formats such as RDF, XML, and OWL. In addition, it supports building and visualizing ontologies as interactive diagrams or trees. Finally, the application allows for integrating ontologies with the R Language [46], in which quantitative and qualitative analysis can be performed.

The taxonomy implements all trace links across the RE, MDE, and SE domains. It shows the relationships between all trace links in these domains. For instance, the diagram in Fig. 2 depicts partial classification of MDE trace links. On the top of the diagram, we can see the root of all elements in the taxonomy which is represented by the word "*thing*" which in turn is connected to *a trace-link* class. A *trace-link* is a general type that has a sub-type *mde-link*. Following this path we reach the leaf nodes such as *Usage*, *Is-A*, and *Export*, which implies those nodes cannot have subtypes. Figure 3 shows an excerpt of the trace links taxonomy, we could not provide the

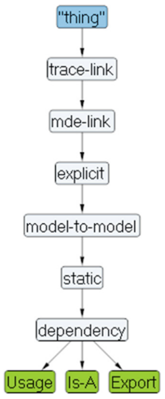


Fig. 2. The leaf and type.

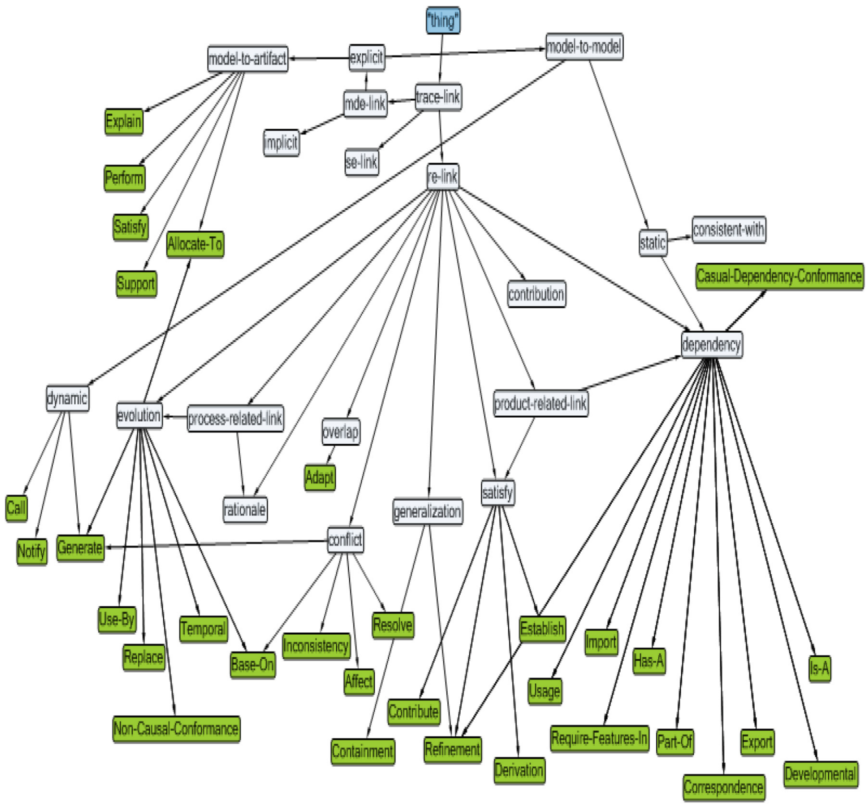


Fig. 3. The trace links taxonomy (excerpt), source [1].

complete taxonomy here since it occupies a big space and it is very hard to visualize all the trace links connections. However, we were, able eliminate the existing redundancies across the RE, SE, and MDE domains. For instance, the *dependency* trace link and some of its leaves (e.g., Refinement) are duplicated in RE and MDE classifications, our taxonomy removes those redundancies by linking the *dependency* to re-link and mde-link, see Fig. 3. Also, at the top left corner of the diagram of Fig. 3., the *Allocate-to* trace link belongs to the *model-to-artifact* and *evolution* categories, we defied it only one in the taxonomy.

6 Taxonomy Validation

The validation of the taxonomy involves the definition of validation criteria, the validation of the taxonomy requirements proposed in Sect. 3, and proposing test cases to be validated through a case study. In addition, we use validation by construction whenever necessary to support our argument.

6.1 Validation Criteria

The taxonomy can be validated by (a) ensuring that it satisfies the taxonomy requirements that we proposed in Sect. 3 in order to resolve the issues about existing classifications, and (b) it can accommodate any traceability problem.

Regarding part (a), we proposed the following validation criteria in order to ensure that all requirements can be satisfied:

- TaxCr 1. Trace Links redundancy. This criterion validates TRQ 7 which states whether a trace link exists more than once in the taxonomy. It should only appear once.
- TaxCr 2. The capability of the taxonomy to accommodate the classification of trace links related to RE, MDE, and SE. This criterion validates TRQ 8 and TRQ 9.
- TaxCr 3. Consistency. This Criterion validates also TRQ 10 whether a trace link has only one definition.
- TaxCr 4. Extensibility and maintainability. This criterion validates TRQ 11 which states how easy a trace link or a property can be added to the taxonomy without changing its design.
- TaxCr 5. Tool Support. This criterion validates TRQ 12 and TRQ 13 which define whether the taxonomy data can be saved, exported to other formats or applications, or performing queries about it.

6.2 Requirements Validation

We validated the requirements of the taxonomy using the validation criteria of Sect. 6.1. The following results are obtained which indicate that the taxonomy meets its requirements.

- Requirement TRQ 1 is verified by construction. Recall TRQ1 is about the specifications of trace link semantics for various artifact types, from different domains and at different levels of granularity. We specified a set of properties for each trace link that define its usage or purpose, its name, type, and definition. These properties provide a semantic for each trace link. Moreover, we established relationships between trace links that exist in more than one domain, these relationships can be used to reason about each trace link. In addition, the URI for each trace link can be used to link artifacts at different levels of granularity: a URI can refer to a single model element (e.g., an operation in a class of a UML diagram, a block in a block diagram), an aggregate of model elements (e.g., a class in a class diagram), a model or document (e.g., a standard document PDF).
- Requirement TRQ 2 is verified by checking whether the taxonomy can link artifacts from different domains and can address the needs of different users. The taxonomy integrates all, at least all those we have identified from a systematic search in the literature and therefore from prominent literature, trace link types that can relate artifacts produced during system analysis, design, coding, and testing, and during model-to-model transformations.

- Requirement TRQ 3 is verified thanks to TaxCr 1 and TaxCr 2. Recall TRQ 3 is about the specification of trace links that are not redundant and that can relate different domains. This requirement is also verified by construction. Indeed, we designed the taxonomy such that any trace link type is specified only once to prevent trace link redundancy. At the same time, our design allows for a trace link to be related to more than one domain. For instance, the *Is_A* trace link in Fig. 3. (Lower right corner) appears only once in the taxonomy, however, it is of type dependency that belongs to the MDE and RE types of traceability data.
- Requirement TRQ 4 is verified thanks to TaxCr 3, which means the requirement is verified by construction. Recall TRQ 4 is about flexibility. The flat hierarchy nature of the taxonomy allows inserting any new trace link type in the taxonomy without the need for reorganizing the taxonomy structure. The new specification can be added anywhere in the RDF code. The same argument applies when we want to add new properties for trace link types; the added properties will apply to all trace links in the taxonomy.
- Requirement TRQ 5 is verified thanks to TaxCr 4 that refers to the use of the URI. Recall that TRQ5 is about portability as well as local/remote access. This was our main purpose for using the RDF. Any trace link in the taxonomy can be referenced to by using its *URI* as shown in Fig. 1. The taxonomy URI references the taxonomy itself and each trace link within the taxonomy can be referenced by using its own URI. Line 10 in Fig. 1 is a good example.
- Requirement TRQ 6 is verified thanks to TaxCr 5. Recall TRQ 6 is about a universal format so the solution can apply to various contexts and is not tied to any tooling support. The taxonomy data can be exported in many universal formats such as HTML, XML, and OWL. In addition, the OWL provides not only data serialization but also verification for the taxonomy semantics.

6.3 Case Study

We describe here a case study from the aviation industry, specifically the Slat/Flap in Airbus aircrafts. We first discuss one important requirement for the Slat/Flap and how it is refined. Then we summarize software development and modeling assumptions for the construction of the Slat/Flap system. We then show how our taxonomy can be used in this context.

Requirement(s). Slats and Flaps are retracted or extended during an aircraft flying and landing. This involves synchronization between various software and hardware components in the aircraft in order to have a safe flight. When a pilot changes the Slat/Flap lever to a certain position, a Command Sensor Unit (CSU) converts the selection into a set of discrete electrical signals. The signals are dispatched and validated by each Slat/Flap Control Computer (SFCC). Accordingly, each SFCC sends a signal independently to the associated hydraulic solenoid on the Power Control Unit (PCU) to allow the hydraulic motors to change the Slat/Flap to the new angle [47].

In A320 and A321 aircraft models, each SFCC produces slightly different outputs as a result of their design by separate vendors. The ambiguity between the outputs of the two SFCC's mandates traceability for the Slat/Flap requirements through all phases of development in order to find the cause of the outputs variations.

The original Slat/Flap requirement for the A321 aircraft states that: "Slat retraction shall be inhibited at a high angle of attack". The development of SFCC's that satisfy this requirement is split into two lanes developed independently by two teams that have no interaction with each other. Moreover, the data required to develop each lane is obtained from different sources. As a result, an inconsistency in the control signals of the two SFCC lanes may occur.

The main Slat/Flap requirement is refined at different levels of granularity, and the description of the main requirement and its refinements span over several documents.

The main requirement and its refinements are summarized in Table 5.

Table 5. Slat/Flap requirements.

Req#	Description
RQ1	*States the main Slat/Flap requirement that is "Slat retraction shall be inhibited at high angle of attack"*
RQ2	*Refines **RQ1** by adding more conditions related to the* high Angle of attack (Alpha-lock), *and the* Low level of attack *(Speed Baulk)*
RQ3	*Shows the rationale about the Slat Alpha-Lock/Speed-Baulk function by expressing the conditions that lead to different function states: disabled, engaged, and reset*
RQ4	*Refines **RQ3** by specifying the values of the Computed Air Speed (CAS) and Computed Angel of Attack (CAoA) for the disabled, engaged, and reset states*
RQ5	*Provides the input/output signals for Slat/Flap retraction in **RQ1**, which are specified by the Alpha-Lock/Speed-Baulk function in **RQ2***

Software Development Process and Modeling. We assume that the IBM DOOR requirement management system is used to store the main requirement and its refinements. Moreover, we assume that these requirements are realized by different UML and Simulink models. More over these requirements are realized by different UML diagrams which include an Activity diagram, a State Machine diagram, and a Component diagram. In addition, we use the Simulink block diagram to model the input/output signals of the Alpha-Lock/Speed-Baulk function. In this case study, the Activity diagram represents the different activities performed by a unit called Air Data/Inertial Reference Unit (ADIRU) and the Slat/Flap Control Computers (SFCC) as stated in **RQ2**. The ADIRU supplies the Corrected Angle of Attack (CAoA) and the Computed Airspeed (CAS) data to the SFCC. The SFCC uses this data to prevent Slat retraction when the CAoA is too large, or the CAS is too low. The Component diagram shows the relationship between the various physical aspects that realize the Slat/Flap requirement and its refined ones. The components are interfaced with each other through different ports. The signal flow diagram provides the input-output signals for the Alpha-Lock/Speed-Baulk function. We assume the signal flow is represented by a block diagram in *Simulink*. The diagram represents the main blocks represent the Command Selection Unit (CSU) which receives the pilot selection as input, converts it to discrete signals and output the signals to the two Slat/Flap control computers, SFCC1 and SFCC2 simultaneously.

Trace Links Identification. We identified several types of trace links between the artifacts of the Slat/Flap problem. They are summarized in Table 6. As indicated by the table, we first established traceability links between requirements, specifically between the main Slat/Flap requirement and its refinements: these are indicated by the first eight rows in the table. For instance, we set a link between requirement RQ1 and RQ2; as per the discussion about requirement RQ1 and its refinements, RQ2 is a refinement of RQ1; as per our taxonomy, the leaf type is therefore *Refine* trace links that belongs to *evolution* link (subtype), which is an *re-link* type (main type). The other requirement-to-requirement links are of the same kind: *re-link/evolution/Refine*; which is in line with the discussion we reported about requirement(s).

We then establish traceability links between diagrams and requirements: traceability links 9–12 in the table. Indeed those diagrams are trace to their requirements, the semantic of the trace link here is *Satisfy*.

Last, we establish traceability links between diagrams: traceability links 13–15 in the table. The traceability in this case exist between either heterogeneous or homogeneous models. The model-to-model relationship in lines 13 and 14 exist between two heterogeneous models (i.e., Simulink and UML models) that don not conform to the same metamodel. We identify the trace links between those two models as follows: first, at higher level of abstraction, the trace link type is *se-link* (i.e., it is not an MDE or RE type). Second, since the two models don not conform to the same metamodel, this means the trace link type is *Vertical*. Third, the two models are modeled within the same system, this implies the trace link type between them is *Intra*. Finally, the two models represent a description of the system at the same level (i.e., during the design phase), which implies a *Micro* trace link type. We classify the link between the artifacts in the latter case as *se-link*, specifically *Intra-Micro* trace link, since the two models are within the same system (i.e., Intra), and within the same phase (i.e., Micro). Indeed this is horizontal traceability since the two models are UML models that conform to the same metamodel.

We provided in Table 6 three different types of trace links classifications based on the granularity of the artifacts and the user needs. As mentioned earlier, at higher level of abstraction, the user can chose the *re-link*, *mde-link*, or the *se-link*. If the user wants to be more specific about the semantic of the trace link, he/she can use one of the types provided in the last column of Table 6. For instance, at a high level of abstraction, the trace link between RQ1 and RQ2 is *re-link* since we have two requirements are linked together at the elicitation phase. However, in order to specify exactly how RQ1 and RQ2 are related to each other, the user can chose the *Refine* trace link to indicate that RQ2 is a refinement RQ1.

The test cases in Table 6 are configured with the trace links taxonomy in order to check whether the taxonomy can satisfy those test cases. Our findings show that all test cases have passed. The taxonomy provides all the trace links needed to link those artifacts.

Table 6. Possible cases for relating the artifacts produced by the Slat/Flap requirement.

Case no.	Source artifact	Target artifact	Trace link		
			Main type	Subtype	Leaf-type
1	**RQ1**	**RQ2**	re-link	Evolution	Refine
2	**RQ1**	**RQ3**	re-link	Evolution	Refine
3	**RQ1**	**RQ4**	re-link	Evolution	Refine
4	**RQ1**	**RQ5**	re-link	Evolution	Refine
5	**RQ2**	**RQ3**	re-link	Evolution	Refine
6	**RQ3**	**RQ4**	re-link	Evolution	Refine
7	**RQ4**	**RQ5**	re-link	Evolution	Refine
8	Activity diagram	**RQ2**	mde-link	Model-to-artifact	Satisfy
9	State machine diagram	**RQ4**	mde-link	Model-to-artifact	Satisfy
10	Component diagram	**RQ2**	mde-link	Model-to-artifact	Satisfy
11	Block diagram	**RQ2**	mde-link	Model-to-artifact	Satisfy
12	Block diagram	Component diagram	se-link	Vertical	Intra-micro
13	Block diagram	Activity diagram	se-link	Vertical	Intra-micro
14	Activity diagram	Component diagram	se-link	Horizontal	Intra-micro

7 Conclusion and Future Work

Existing trace links classifications have some drawbacks that affect their usage in relating artifacts from different domains (e.g., software, system, mechanical). We have proposed a trace link taxonomy that can be used as part of a traceability framework to capture traceability information among artifacts. The taxonomy is built be linking local and external resources (taxonomy elements) to form a flat hierarchical structure. We implemented the taxonomy using the Resource Description Framework, which allows for referencing elements using their URI. This technique provides many advantages over other classical techniques such as allowing the definition of relational or hierarchical structures. It offers interoperability and portability of data among different platforms. Moreover, data are readable by human and machines as well, and it can be transformed into several textual and graphical formats.

The trace link taxonomy we describe coalesces elements and concepts of taxonomies previously defined for different domains: specifically requirement engineering, system engineering and model driven engineering. When unifying taxonomies we take good care and remove redundant concepts. Our taxonomy provides a precise semantic for each trace link by allowing the user to specify its name and usage, among other properties. Moreover, it offers trace links editing, filtering and searching. Furthermore, the taxonomy is a knowledge base from which we can reason about its elements and infer relationships and provide statistics about trace links. Our taxonomy is created to satisfy specific requirements, is verified, and validated on a case study from the aerospace domain.

As a future work, this taxonomy can be extended by adding more trace link properties and trace links types, though we believe that thanks to our systematic literature review, we likely have collected and integrated most of that information already. Researchers in software engineering and traceability are invited to build upon this taxonomy. We believe this taxonomy can be used as a base for standardizing trace links semantics. Future work should, of course, consider using our taxonomy so as to increase our confidence that it addresses needs of various stakeholders dealing with traceability, as we expect it does.

References

1. Mustafa, N., Labiche, Y.: Employing linked in building a trace links taxonomy. In: International Conference of Software Technologies, Spain (2017)
2. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Trans. Softw. Eng. **27**(1), 58–93 (2011)
3. Gotel, O., Finkelstein, A.: An analysis of the requirements traceability problem. In: 1st International Conference on Requirements Engineering, Utrecht, The Netherlands (1994)
4. Paige, F., et al.: Building model-driven engineering traceability classifications. In: European Conference on Model Driven Architecture - Traceability Workshop, Berlin, Germany (2008)
5. Mason, P., et al.: Meta-modelling approach to traceability for avionics: a framework for managing the engineering of computer based aerospace systems. In: 10th IEEE International Conference on Engineering of Computer-Based Systems. IEEE, Huntsville (2003)
6. Spanoudakis, G., et al.: Rule-based generation of requirements traceability relations. Syst. Softw. **72**(2), 105–127 (2004)
7. Spanoudakis, G., Zisman, A.: Software traceability: a road map. In: Chang, S.K. (ed.) Handbook of Software Engineering and Knowledge Engineering, pp. 395–428 (2005)
8. Xu, P., Ramesh, B.: Supporting workflow management systems with traceability. In: 35th Annual Hawaii International Conference on System Sciences. IEEE, Hawaii (2002)
9. Pohl, K.: PRO-ART: enabling requirements pre-traceability. In: 2nd IEEE International Conference on Requirements Engineering. IEEE Computer Society (1996)
10. Alexander, I.: Semi automatic tracing of requirement versions to use cases – experience and challenges. In: 2nd International Workshop on Traceability in Emerging Forms of Software Engineering, Canada (2003)
11. Riebisch, M., Philippow, I.: Evolution of product lines using traceability. In: Workshop on Engineering Complex Object-Oriented Systems for Evolution, Florida (2001)
12. Object Management Group: Unified Modeling Language (UML) (2015). http://www.uml.org/. Accessed 10 May 2015
13. OMG, O.M.G.: OMG systems modeling language (2014). http://www.omgsysml.org/. Accessed 10 June 2014
14. Maletic, J.I., et al.: Using a hypertext model for traceability link conformance analysis. In: 2nd International Workshop on Traceability for Emerging Forms of Software Engineering, Canada (2003)
15. Pinheiro, F.A.C., Goguen, J.A.: An object-oriented tool for tracing requirements. IEEE Softw. **13**(2), 52–64 (1996)
16. Gotel, O., Finkelstein, A.: Contribution structures. In: 2nd International Symposium on Requirements Engineering, IEEE (1995)

17. Mustafa, N., Labiche, Y.: The need for traceability in heterogeneous systems: a systematic literature review. In: IEEE International Computers, Software and Applications Conference, Italy (2017)
18. Constantopoulos, P.J.M., Mylopoulos, Y., Vassiliou, Y.: The software information base: a server for reuse. Int. J. Very Large Data Bases **4**(1), 1–43 (1993)
19. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering, in EBSE Technical report (2007)
20. Letelier, P.: A framework for requirements traceability in UML-based projects. In: 1st International Workshop on Traceability in Emerging Forms of Software Engineering (2002)
21. Mustafa, N., Labiche, Y.: Modeling traceability for heterogeneous systems. In: 10th International Conference on Software Engineering and Applications. SCITEPRESS, Colmar (2015)
22. IEEE: IEEE Standard Glossary of Software Engineering Terminology. In: IEEE Standard Glossary of Software Engineering Terminology, I.S. board Editor, New York (1990)
23. Cleland-Huang, J., Gotel, O., Zisman, A. (eds.): Software and Systems Traceability. Springer, Heidelberg (2014). https://doi.org/10.1007/978-1-4471-2239-5
24. Gotel, O., et al.: Traceability fundamentals. In: Cleland-Huang, J., Gotel, O., Zisman, A. (eds.) Software and Systems Traceability, pp. 3–22. Springer, Heidelberg (2012). https://doi.org/10.1007/978-1-4471-2239-5_1
25. Ramesh, B., Edwards, M.: Issues in the development of a requirements traceability model. In: IEEE International Symposium on Requirements Engineering (1993)
26. Aizenbud-Reshef, N., et al.: Model traceability. IBM Syst. J. Model Driven Softw. Develop. **45**(3), 515–526 (2006)
27. Mustafa, N., Labiche, Y.: Toward traceability modeling for the engineering of heterogeneous systems. In: International Conference on Model Driven Engineering and Software Development, Angers, Loire Valley, France (2015)
28. Dick, J.: Rich traceability. In: 1st International Workshop on Traceability for Emerging forms of Software Engineering (2002)
29. Mohan, K., Ramesh, B.: Managing variability with traceability in product and service families. In: 35th Annual Hawaii International Conference on System Sciences. IEEE, Hawaii (2002)
30. Grammel, B.: Automatic generation of trace links in model-driven software development. Fakultät Informatik, Technische Universität Dresden (2014)
31. Olsen, G.K., Oldevik, J.: Scenarios of traceability in model to text transformations. In: Akehurst, D.H., Vogel, R., Paige, R.F. (eds.) ECMDA-FA 2007. LNCS, vol. 4530, pp. 144–156. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72901-3_11
32. Paige, R.F., et al.: Rigorous identification and encoding of trace-links in model-driven engineering. Softw. Syst. Model. **10**(4), 469–487 (2011)
33. Lucia, A.D., Fasano, F., Oliveto, R.: Recovering traceability links in software artifact management systems using information retrieval methods. ACM Trans. Softw. Eng. Methodol. **16**(4), 13 (2007)
34. Rummler, A., Grammel, B., Pohl, C.: Improving traceability in model-driven development of business applications. In: European Conference on Model Driven Architecture - Traceability Workshop (2007)
35. Knethen, A.: Automatic change support based on a trace model. In: 1st International Workshop on Traceability in Emerging Forms of Software Engineering, Edinburgh (2002)
36. Filho, G.C., Zisman, A., Spanoudakis, G.: Traceability approach for i* and UML models. In: International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, Portland (2003)

37. LindVall, M., Sandahl, K.: Practical implications of traceability. Softw. Pract. Exp. **26**(10), 1161–1180 (1996)
38. W3C: Resource Description Framework (2016). https://www.w3.org/RDF/. Accessed 15 Oct 2016
39. Kozlenkov, A., Zisman, A.: Are their design specifications consistent with our requirements? In: IEEE Joint International Conference on Requirements Engineering. IEEE (2002)
40. OMG, O.M.G.: Unified Modeling Language (2014). http://www.uml.org/. Accessed 10 July 2014
41. OMG, O.M.G.: Systems Modeling Language (2014). http://www.omgsysml.org/. Accessed 10 June 2014
42. Roques, P.: Modeling requirements with SysML. In: Requirement Engineering Magazine. IREB (2015)
43. Miller, L., Brickley, D.: FOAF (2016). http://www.foaf-project.org/. Accessed 3 Nov 2016
44. Dumbill, E.: Description of a Project (2016). http://lov.okfn.org/dataset/lov/vocabs/doap. Accessed 3 Nov 2016
45. Cognitum: Fluent Editor 2015 (2017). http://www.cognitum.eu/semantics/FluentEditor/. Accessed 2 Feb 2017
46. R Foundation: The R project for statistical computing (2017). https://www.r-project.org/. Accessed 2 Feb 2017
47. Bretschneider, M., et al.: Model-based safety analysis of a flap control system. Int. Counc. Syst. Eng. **14**(1), 246–256 (2004)

A Machine Learning Approach for Game Bot Detection Through Behavioural Features

Mario Luca Bernardi[1], Marta Cimitile[2(✉)], Fabio Martinelli[3], and Francesco Mercaldo[3]

[1] Giustino Fortunato University, Benevento, Italy
[2] Unitelma Sapienza, Roma, Italy
marta.cimitile@unitelma.it
[3] Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy

Abstract. In the last years, online games market has been interested by a sudden growth due to the birth of new gaming infrastructures that offer more effective and innovative services and products. Simultaneously to the diffusion of on line games, there was an increasing use of game bots to automatically perform malicious tasks. Game bots users aim to obtain some rewards by automating the most tedious and prolonged activities arousing the disappointment of the game community. Therefore, the detection and the expulsion of game bots from the game environment, become critical issues for the game's developers that want to ensure the satisfaction of all the players. This paper describes an approach for the game bot detection in the online role player games consisting to distinguish between game bots and human behavior and based on the adoption of supervised and unsupervised machine learning techniques. These techniques are used to discriminate between users and game bots basing on some user behavioral features. The approach is applied to a real-world dataset of a popular role player game and the obtained results are encouraging.

Keywords: Game bot · Machine learning · Cluster analysis Game bot detection · Security · Testing

1 Introduction

The worldwide games market is largely grown in the last years (the revenue increases of 7.8% in the first quarter of 2017 with respect to the year before)[1] due to the development of innovative online gaming platforms and the availability of a huge and high-quality games offering. As a matter of fact, an increasing interest

[1] https://newzoo.com/solutions/standard/market-forecasts/global-games-market-report/.

E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 114–134, 2018.
https://doi.org/10.1007/978-3-319-93641-3_6

has been pointed to the on line games, that thanks to the diffusion of Internet network, are become very popular. Differently from traditional games, on line games are partially or primarily played through the Internet network or another computer network [1]. These games are made available through modern gaming platforms that include PCs, consoles and mobile devices, and offer many genres (i.e., first-person shooters, strategy games and massively multiplayer online role-playing games) [2]. Moreover, online games can have a different design (from simple text-based environments to more complex graphics and virtual worlds) [3] and attract players from a variety of ages, nationalities, and occupations [4,5]. However, given their diffusion and their characteristics, on line games represent an appealing scenario for attackers to perpetrate illegal activities in the online world [6]. They are interested to perform illegal activities for several reasons: get economic gain (cyber assets can be changed into real currency) [7], capture reserved information about the other players or became popular in the game community. Several malicious tasks are performed by using game bots able to repeat continuously a given activity in the online game. Game bots consist of an artificial intelligent software that simulates human behavior in a video game [8]. Since they automatize playing actions, game bots can earn more cyber assets than human users because they can play at a full time without any break. Moreover, game bots cause harm to the human users because they consume game resources reducing the player opportunities: for example, game bots defeat all monsters and harvest the available items before the human users. Basing on the above considerations, we can conclude that game bots damage the game provider reputation and reduce the game's lifecycle [9]. This paper extends and explores a method for game bots detection in a Multiplayer Online Role-Playing Game (MMORPG environment) proposed in [10,11]. MMORPGs are a kind of online game were a large number of players interact with one another within a virtual world. The method uses some features (i.e., Player Information, Player Actions, Group Activities, Social Interaction Diversity and Network Measures) to distinguish game bots from human players behavior. This discrimination is performed by using some classifiers built thought supervised and unsupervised (i.e., cluster analysis) machine learning techniques. The effectiveness of our proposed approach is evaluated in a real context consisting of a dataset obtained from the operation of Aion: The Tower of Eternity[2] game. It is a MMORPG fantasy free-to-play popular game where all the operations performed by real players and game bots are identified and labeled by the game company. The rest of the paper is organized as follows: the next section provides an overview of the related work. In Sects. 3 and 4 the proposed features and the detection techniques are illustrated. In Sect. 5, authors evaluate if it is possible to detect game bots by using the set of the proposed behavioral features and evaluate the effectiveness of the proposed approach to discriminate human and botnet behaviors in a real MMORPG game. Section 5 shows the results of the evaluation finally, conclusion and future works are given in the last section.

[2] https://en.aion.gameforge.com/website/.

2 Related Work

A great number of approaches are recently proposed on the topic of game bots detection. Several methods adopt data mining techniques to analyze the log data extracted by the game server. These approaches are favorite by the game service providers because they allow to detect and block the game bots without any interference with the game user playing and without the deployment of additional software on the client-side. Some server-side approaches are focused on the analysis of users interactions and social activity [12] assuming that game bots and human player have different game style discriminating them (these approaches are also called behavioral approaches). For example, in [13], the social behaviors of both human and game bots are modeled and compared through the use of behavioral features. In [14], some thresholds are fixed to discriminate the different behaviors of the game bots and the human players basing on their activities. In [15], the bot detection is performed by analyzing the sequences of window events (described as a set of attributes) produced by the game players. Learning algorithms are used to distinguish and classify human from game bots event sequences. Authors, in [16,17] describe a manifold learning approach consisting to analyze the avatars movement trajectories to distinguish game bots from human players (they are assumed to have different movement trajectories). Moreover, in [18] authors compare the traffic generated by the game bots versus human players to perform game bots detection. A specific focus on the MMORPG environment is in [19] where the characteristics of a MMORPG environment are analyzed and a method for detecting GFGs (i.e., gold farming groups) is explored. MMOPG is also considered in [20]. Here, authors perform action sequence analysis on a server-side account theft detection system to protect game users from malicious players. Moreover, several approaches are based on the analysis of behavioral approaches [9,21–24]. For example, [13] uses some features to capture the social behavior of the game players assuming that game bots and humans tend to form their social network in different ways. The behavioral approaches analyze a limited number of features (one or two behavioral features) having, as consequence, a high dependence of the approach from the single game domain. This limit is discussed and faced in [25], where authors propose a new approach based on a larger set of features. It uses both game and player-related features similar to our proposed approach. With respect to [25], in our proposed approach the obtained results show a higher precision in the game bot detection. Moreover, differently from [25], in this work, a feature selection step is provided in order to improve the performance in the real environments. This step aims to select the most discriminative features (they will become input for the classification step) between human users and game bot and the time employed to build the classifiers is evaluated as the maximum time interval to detect a game bot behavior.

3 Background

In order to evaluate the effectiveness of the behavioral feature sets to discriminate between human and game bot, we use supervised and the unsupervised [26] machine learning approaches. Indeed, machine learning tasks are typically classified into these two categories, depending on the nature of the learning available to a learning system [27,28]. Supervised learning is the machine learning task of inferring a function from the labeled training data. In this case, the training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (i.e., the behavioral feature sets) and the desired output value (i.e., the game bot of the human label). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new features without a label. The optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. Differently, the unsupervised machine learning (also called cluster analysis) is the machine learning task of inferring a function to describe hidden structure from "unlabeled" data (a classification or categorization is not included in the observations). For this reason, using this approach is possible to infer the human and the game bot label without previous knowledge about the nature of the feature sets considered (i.e., these algorithms do not require the label). Relating to supervised machine learning classification, we consider six different algorithms of classification: J48, DecisionStump, HoeffdingTree, RandomForest, RandomTree and REPTree [29,30]. The supervised used algorithms are listed in the following:

– *J48* [31]: It is an open source Java implementation of the C4.5 decision tree algorithm. It is a statistical classifier based on id3 algorithm [32]. Basically, it computes the potential information for every considered attribute, given by a test on the attribute and the gain in information is calculated that would result from a test on the attribute.
– *DecisionStump* [33]: A decision tree-based model with one internal root node immediately connected to the terminal nodes. The prediction is allowed by a decision stump basing on the value of just one input feature.
– *HoeffdingTree* [34]: An incremental, anytime decision tree induction algorithm learning from massive data streams. Basing on the assumption that the distribution generating examples does not change over time, it often uses a small sample to choose an optimal splitting attribute.
– *RandomForest* [35]: It operates by constructing, at training time, a multitude of decision trees and outputting the class that is the mode (the most frequent value appearing in a set of data) of the classes of the individual trees.
– *RandomTree* [36]: It constructs a tree containing some randomly chosen attributes at each node. No pruning is performed.
– *REPTree* [37]: It builds a decision tree using information gain/variance and reduces errors using reduced-error pruning. The values are ordered for numeric attributes once and missing values are recovered with by splitting the corresponding instances into pieces.

With regards to the unsupervised machine learning classification, we consider the following six cluster analysis algorithms: SimpleKMeans, FarthestFirst, FilteredClusterer, MakeDensityBasedClusterer, Cobweb, DBScan.

- *SimpleKMeans* [38]: Cluster data using the k-means algorithm with the Euclidean distance. The k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.
- *FarthestFirst* [39]: The farthest-first traversal of a bounded metric space computes a sequence of points in the space, where the first point is selected arbitrarily and each successive point is as far as possible from the set of previously-selected points.
- *FilteredClusterer* [40]: Algorithm for running an arbitrary clusterer on data that has been passed through an arbitrary filter. Like the clusterer, the structure of the filter is based exclusively on the training data and test instances will be processed by the filter without changing their structure.
- *MakeDensityBasedClusterer* [41]: Algorithm for wrapping a clusterer to return a distribution and density. It fits normal distributions and discrete distributions within each cluster produced by the wrapped clusterer.
- *Cobweb* [42]: It is an incremental system for hierarchical conceptual clustering. This algorithm incrementally organizes observations into a classification tree. Each node in a classification tree represents a class (concept) and it is labeled by a probabilistic concept that summarizes the attribute-value distributions of objects classified under the node. This classification tree can be used to predict missing attributes or the class of a new object.
- *DBScan* [43]: The Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm. It is a density-based clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers the points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.

4 The Method

As discussed in Sect. 1, our proposed approach aims to discriminate game bots from human players basing on their different behavior in the context of MMORPG games. For this reason, we propose a set of behavioral features and we evaluate their capability to discriminate human users and game bots. The considered features were extracted and computed by a real game company (as described in [9]). They are grouped in the following categories: Player Information (PI), Player Actions (PA) (they describe the player's characteristics), Group Activities (GA), Social Interaction Diversity (SID) and Network Measures (NM). The first two features groups describe players characteristics, while the other features categories are referred to the game characteristics. Table 1 shows the considered features for each category listed above.

Table 1. The features involved in the study with the correspondent category [44].

Category	Features
Player Information	$\mathrm{PI}_1, \mathrm{PI}_2, \mathrm{PI}_3, \mathrm{PI}_4$
Player Actions	$\mathrm{PA}_1, \mathrm{PA}_2, \mathrm{PA}_3, \mathrm{PA}_4, \mathrm{PA}_5, \mathrm{PA}_6, \mathrm{PA}_7, \mathrm{PA}_8, \mathrm{PA}_9, \mathrm{PA}_{10}$
Group Activities	$\mathrm{GA}_1, \mathrm{GA}_2$
Social Interaction Diversity	SID_1
Network Measures	$\mathrm{NM}_1, \mathrm{NM}_2, \mathrm{NM}_3, \mathrm{NM}_4, \mathrm{NM}_5, \mathrm{NM}_6, \mathrm{NM}_7, \mathrm{NM}_8, \mathrm{NM}_9$

Looking to the table, the Player Information (PI) features describe how the players perform some game tasks. The main assumption is that the PI features analysis shows a gap between the values of these features for game bots and for human users. The considered PI features are the followings: login frequency (PI_1), play time (PI_2), game money (PI_3) and a number of IP address (PI_4). Similarly, the PA features are introduced to investigate whether the player actions can reflect the differences between game bots and human users. For instance, game bots are often connected to the game at a full time and can play for 24 consecutive hours, differently from human users, that typically are not able to play during several time-windows (i.e., work time, sleeping time). This allows them to reach a certain rank level and obtain more powers to fight the enemies effectively. Moreover, more player kill points can be acquired by defeating players of opposing factions. Player kill points can be used to purchase various items from vendors and to determine a player's rank within the game world. In the Aion game[3], the more player kill points a player has, the higher is the rank of the player. The high ranking player can feel a sense of accomplishment. The attackers are interested to obtain game powers because they can resell these powers to the other game users[4]. Finally, the game bots sit more frequently than human users in order to recover health and many points. The considered PA features are: sitting (PA_1) (i.e., an action taken by players to recover their health), earning experience points (PA_2), obtaining items (PA_3), earning game money (PA_4), earning player kill points (PA_5), harvesting items (PA_6), resurrecting (PA_7), restoring experience points (PA_8), being killed by a non-player and/or player character (PA_9), and using portals (PA_{10}). The GA features are introduced to evaluate the different behavior of game bots and human players with respect to their participation in the group activities. The rationale behind the GA feature is that there is a gap between the values of the social features of game bots and those of human users because game bots do not attempt to social as humans. For example, game bots are not interested in the activities that allow increasing the socialization among the player's community but they only perform social activities that are required to obtain game advantages. As matter of fact, as a consequence of the game bot protract play time, we expected

[3] https://sites.google.com/a/hksecurity.net/ocslab/Datasets/game-bot-detection.
[4] http://www.geek.com/games/.

that the GA category is different between game bots and human users. The GA category includes the average duration of party play (GA_1) and the number of guild activities (GA_2). For example, looking to the GA_1, party play is a group play formed by two or more players in order to undertake quests or missions together. Party play aims to complete difficult quests by collaboration and enjoy socialization. Differently, from other players, game bots perform party play, with the only goal to acquire game money and items faster and more efficiently in order to obtain powers. For this reason, we expected that the value obtained for GA_1 is different between game bots and human users. The concept of party play is also useful to understand the SID_1 features defined as the entropy of party play. Game bots concentrate only on particular actions, whereas human users execute multiple tasks as needed to thrive in the online game world. Finally, the NM category regards the player's social interaction network that can be represented as a graph with characters as the nodes and interactions between them as the edges. An edge between two nodes (players) in this graph may, for example, highlight the transfer of an item between the two nodes. The features of NM include the degree centrality (NM_1), betweenness centrality (NM_2), closeness (NM_3), eigenvector centrality (NM_4), eccentricity (NM_5), authority (NM_6), hub (NM_7), PageRank (NM_8), and clustering coefficient (NM_9). The NM category features are explained in Table 2.

5 The Evaluation

The proposed experiment aims to evaluate the effectiveness of the feature vector we propose, with respect to the goals stated in the introduction. More specifically, the capability of the behavioral features to discriminate the game bots by the human user behavior is here evaluated. A classification of the proposed features values is carried out by using several state-of-the-art machine learning classifiers built with the behavioral feature categories we considered. In this study, a real dataset, obtained from the operation of a popular and free available game called Aion, is used. It contains all in-game action logs for 88 days, between April 9th and July 5th of 2010. During this period, 49,739 players attend the game for at least three hours. Among these players, 7702 characters are identified by the game company as game bots. The banned list provided by the game company is used in our evaluation as the ground truth after that each banned user has been vetted and manually verified by human labor and active monitoring. Starting by the log released by the game company, in our dataset we aggregated the values of the features related to the same user and we labeled the feature vectors as "human players" or "game bot". Moreover, in the dataset all the private and personal information of the players are protected by the anonymity. The evaluation consists of some main steps: (i) a comparison of the descriptive statistics of the behavioral features populations; (ii) the hypotheses testing, aiming to verify if the feature categories have different distributions for the populations of game bot and human behavior; and (iii) the classification analysis aimed at assessing whether the behavioral feature categories are able to correctly classify game

Table 2. The features belonging to the Network Measures category with their description [44].

NM category feature	Description
Degree centrality	This features represents the centrality focused on the degree. The more edges an actor has, the more important it is
Betweenness centrality	It counts the number of shortest paths between two nodes on which a given actor resides
Closeness centrality	An actor is considered important if it is relatively close to all other actors. Closeness is based on the inverse of the distance of each actor to every other actor in the network
Eigenvector centrality	Indicates that a given node has a relationship with other valuable nodes. A high eigenvector value for an actor means that a node has several neighbors with high eigenvector values
Eccentricity	The eccentricity of node v is calculated by computing the shortest path between node v and all other nodes in the graph; then the longest shortest path is chosen
Authority	Exhibits a node pointed to by many good hubs
Hub	Exhibits a node that points to many good authorities
PageRank	Assigns a numerical weight to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of "measuring" its relative importance within the set
Clustering coefficient	It quantifies how close neighbors are to being a clique: a clique is a subset of all of the edges connecting pairs of vertices of an undirected graph

bot and human behavior. The classification analysis is performed by using both supervised and unsupervised machine learning algorithms.

5.1 Descriptive Statistics

In this section the box plots of the distribution of game bot and human behavior features are shown in order to demonstrate their differences. Figures 1, 2 and 3 show the box plots for a subset of features belonging to each category considered in the study. Figure 1(a) shows the box plot related to the login frequency time feature (PI_1) between game bot and human users. Since they assume similar values, we can conclude that game bot and human user login with similar frequency. In our opinion, the limit of this feature is that it does not consider the login time but only the login frequency. Figure 1(b) shows the box plot for the play time feature (PI_2). The figure highlights that the distributions between game bots and human users are very different: as matter of fact, while human user presents a small distribution, the game bot one exhibits higher play time

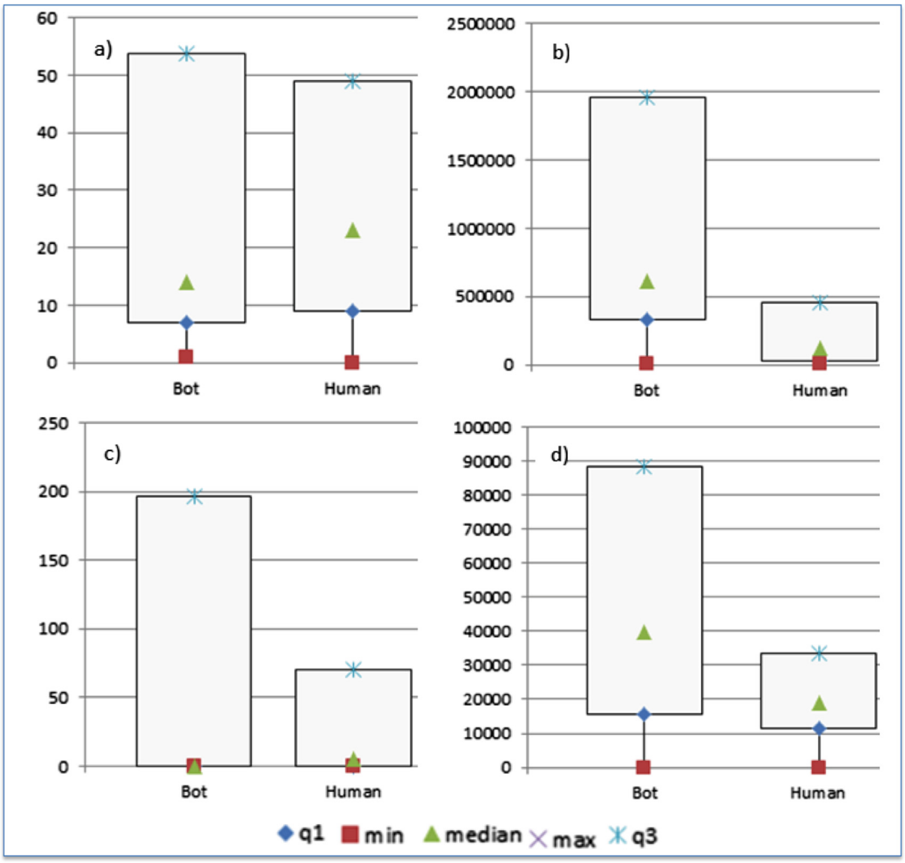


Fig. 1. The box plot relating to the game bot and human distributions for the login frequency feature (a) the play time feature (b) the sitting feature (c) and for the earning experience points feature (d).

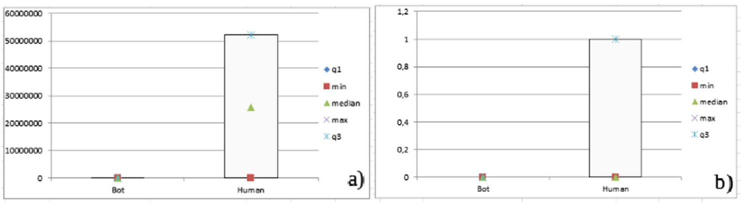


Fig. 2. The box plot relating to the game bot and human distributions for the party play time feature (a) and the guild activities feature (b), belonging to the Group Activities category.

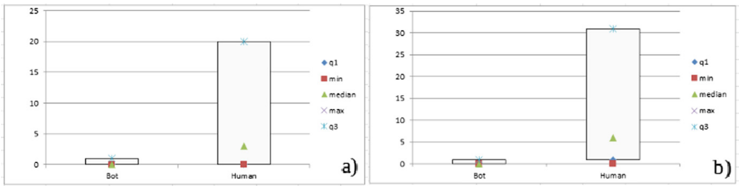


Fig. 3. The box plot relating to the game bot and human distributions for the degree centrality feature (a) and the betweenness centrality feature (b), belonging to the Network Measures category.

values. The reason is that game bot is able to play the game without any break, differently from human users.

Figure 1(c) and (d) show the box plot related to game bot and human distributions for the feature belonging to the Player Actions category. Figure 1(c) shows that the sitting time distribution for game bots seems to be wider if compared with the human users one. This feature is related to the number of rounds that game bots and human users are able to play.

Figure 1(d) shows the box plot related to earning experience points feature (PA_2). This feature evaluates the ability to earn points in order to buy power for the character. The figure shows a wider dimension of game bots distribution with respect to the human users one confirming that the game bots are able to gather more points if compared with human users.

Figure 2(a) shows the box plot related to the party play time feature (GA_1). The distribution obtained for the human user box plots is wider if compared with the game bots one. This happens because game bots usually have not interest to make party play in order to play o socialize with other users: the focus of game bots is only to disturb human users and gather points in order to have character reinforcements.

Figure 2(b) shows the box plot related to the guild activities feature (GA_2). The feature is related to the capacity of a player to perform missions in cooperation with other players. Confirming the previous box plot, game bot has no interest to cooperate in the play, while human user usually needs to play together in order to complete successfully difficult missions.

Figure 4 shows the distributions for the SID_1 feature. The distributions appear very similar, highlighting that both game bots and human users interact with them, for this reason, the considered feature is not discriminative between the observed populations.

Figure 3(a) shows the box plot related to the NM_1 feature. These box plots exhibit that human users present a wider degree centrality if compared with game bots. We conclude that human users have more friends if compared with game bots.

Figure 3(b) shows the NM_2 feature box plot. As in the previous box plots, the human user's distribution is wider than the game bots one. It happens because the human user tries to reach an objective by using the shortest path, differently from game bots that do not consider this.

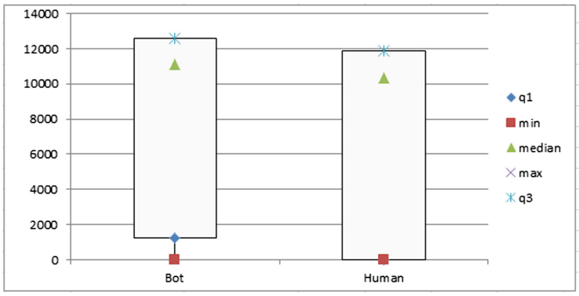


Fig. 4. The box plot relating to the game bot and human distributions for the party play entropy, belonging to the Social Interaction category [44].

5.2 Hypothesis Testing

Basing on the research goals discussed in Sect. 1, the hypotheses testing and the null hypothesis can be the following:

H_1: There are statistically significant differences between the considered features of game bots and human users
H_0: There are no statistically significant differences between the considered features of game bots and human users

The null hypothesis was tested with three different tests in order to enforce the conclusion validity: Mann-Whitney, Kolmogorov-Smirnov Test and Wald-Wolfowitz. For all the tests the p-level is fixed to 0.05. The purpose of these tests is to determine the level of significance, i.e., the risk (the probability) that erroneous conclusions be drawn: since we set the significance level equal to 0.05, we accept to make mistakes 5 times out of 100. The hypothesis testing aims at evaluating if the features present different distributions for the populations of the game bot and human behavioral characteristics with statistical evidence. We assume valid the results when the null hypothesis is rejected by both the tests performed. Table 3 shows the results of hypothesis testing: the null hypothesis H_0 can be rejected for all the features. This means that there is statistical evidence that the feature vector is a potential candidate for correctly classifying between game bot and human behavioral characteristics. This result will provide an evaluation of the risk enforced by the fact that three tests are used and the selected features produce values which belong to two different distributions (i.e., the one related to the game bots and the human users). With the classification analysis, we will be able to establish the accuracy of the features in associating any behavioral feature to the game bot or to the human distribution.

5.3 Classification Analysis

This step consists of building classifiers in order to evaluate the feature vector accuracy to distinguish between game bots and human users. Four metrics were

Table 3. Results of the hypothesis test of the null hypothesis H_0.

Variable	Mann-Whitney	Kolmogorov-Smirnov	Wald-Wolfowitz
PI1	0.000000	$p < .001$	0.000
PI2	0.000000	$p < .001$	0.000
PI3	0.000000	$p < .001$	0.000
PI4	0.000000	$p < .001$	0.000
PA1	0.000000	$p < .001$	0.000
PA2	0.000000	$p < .001$	0.000
PA3	0.000000	$p < .001$	0.000
PA4	0.000000	$p < .001$	0.000
PA5	0.000000	$p < .001$	0.000
PA6	0.000000	$p < .001$	0.000
PA7	0.000000	$p < .001$	0.000
PA8	0.000000	$p < .001$	0.000
PA9	0.000000	$p < .001$	0.000
PA10	0.000000	$p < .001$	0.000
GA1	0.000000	$p < .001$	0.000
GA2	0.000000	$p < .001$	0.000
SI1	0.000000	$p < .001$	0.000
NM1	0.000000	$p < .001$	0.000
NM2	0.000000	$p < .001$	0.000
NM3	0.000000	$p < .001$	0.000
NM4	0.000000	$p < .001$	0.000
NM5	0.000000	$p < .001$	0.000
NM6	0.000000	$p < .001$	0.000
NM7	0.000000	$p < .001$	0.000
NM8	0.000000	$p < .001$	0.000
NM9	0.000000	$p < .001$	0.000

used to evaluate the classification results: Precision, Recall, F-Measure and ROC Area. The precision has been computed as the proportion of the examples that truly belong to class X among all those which were assigned to the class. It is the ratio of the number of relevant records retrieved to the total number of irrelevant and relevant records retrieved:

$$Precision = \frac{tp}{tp + fp}$$

where *tp* indicates the number of true positives and *fp* indicates the number of false positives. The recall has been computed as the proportion of examples that were assigned to class X, among all the examples that truly belong to the class, i.e., how much part of the class was captured. It is the ratio of the number of relevant records retrieved to the total number of relevant records:

$$Recall = \frac{tp}{tp + fn}$$

where *tp* indicates the number of true positives and *fn* indicates the number of false negatives. The F-Measure is a measure of a test's accuracy. This score can be interpreted as a weighted average of the precision and recall:

$$F\text{-}Measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The Roc Area is defined as the probability that a positive instance randomly chosen is classified above a negative randomly chosen. The classifier training is performed by defining T as a set of labeled traces *(M, l)*, where each M is associated to a label $l \in \{B, H\}$ (where B represents the game bot, while H the human user). For each M, a feature vector F $\in R_y$ is build, where y is the number of the features used in training phase ($y = 4$ for the PI category, $y = 9$ for the PA category, $y = 2$ for the GA category, $y = 1$ for the SID category and $y = 9$ for the NM category). In the learning phase, we use a k-fold cross-validation: the dataset is randomly partitioned into k subsets. A single subset is retained as the validation dataset for testing the model, while the remaining $k - 1$ subsets of the original dataset are used as training data. We repeated the process for $k = 10$ times; each one of the k subsets has been used once as the validation dataset. To obtain a single estimate, we computed the average of the k results from the folds. We evaluated the effectiveness of the classification method with the following procedure:

1. build a training set $T \subset D$;
2. build a testing set $T' = D \div$ T;
3. run the training phase on T;
4. apply the learned classifier to each element of T'.

Each classification was performed using 20% of the dataset as training dataset and 80% as testing dataset employing the full feature set. The obtained results are shown in Table 4.

Looking at the table, we can conclude that a precision equal to 0.952 and a recall of 0.953 is obtained by using the J48 algorithm to classify the feature belonging to PI category. Similarly, a precision equal to 0.954 and a recall of 0.955 is obtained with the RandomForest algorithm classifying the feature belonging to PA category while RepTree algorithm shows a precision equal to 0.858 and a recall a 0.882 on the feature belonging to GA category. J48 also gives a precision equal to 0.836 and a recall equal to 0.875 on the feature belonging to SID category. Finally, a precision equal to 0.923 and a recall 0.928 using the RandomForest algorithm classifying the feature belonging to NM category. The table also shows that we obtained the best results (in terms of precision and recall) when classifying the features related to the PI and PA categories.

Table 4. Classification results: Precision, Recall, F-Measure and RocArea for classifying the feature categories, computed with six different classification algorithms [44].

Category	Algorithm	Precision	Recall	F-Measure	Roc Area	Time
PI	J48	0.95	0.951	0.949	0.856	1.97
	DecisionStump	0.942	0.944	0.941	0.823	0.21
	HoeffdingTree	0.941	0.944	0.941	0.872	0.3
	RandomForest	0.952	0.953	0.950	0.895	37.8
	RandomTree	0.917	0.916	0.916	0.814	0.64
	REPTree	0.946	0.948	0.945	0.880	0.93
PA	J48	0.948	0.950	0.947	0.862	7.01
	DecisionStump	0.934	0.936	0.935	0.830	0.58
	HoeffdingTree	0.942	0.944	0.942	0.872	0.94
	RandomForest	0.954	0.955	0.953	0.95	57.23
	RandomTree	0.952	0.952	0.921	0.901	4.01
	REPTree	0.948	0.950	0.948	0.888	3.36
GA	J48	0.858	0.881	0.843	0.764	0.38
	DecisionStump	0.764	0.874	0.816	0.721	0.05
	HoeffdingTree	0.854	0.880	0.839	0.783	0.17
	RandomForest	0.820	0.855	0.833	0.766	17.71
	RandomTree	0.820	0.854	0.833	0.758	0.42
	REPTree	0.858	0.882	0.845	0.784	0.47
SID	J48	0.836	0.875	0.821	0.698	0.37
	DecisionStump	0.764	0.874	0.816	0.690	0.06
	HoeffdingTree	0.826	0.874	0.823	0.710	0.32
	RandomForest	0.805	0.863	0.822	0.688	8.31
	RandomTree	0.805	0.865	0.822	0.682	0.14
	REPTree	0.829	0.874	0.823	0.717	0.11
NM	J48	0.917	0.923	0.917	0.810	24.75
	DecisionStump	0.871	0.884	0.875	0.673	0.99
	HoeffdingTree	0.892	0.903	0.891	0.769	2.68
	RandomForest	0.923	0.928	0.923	0.858	42.42
	RandomTree	0.886	0.887	0.887	0.751	0.69
	REPTree	0.917	0.923	0.917	0.845	5.5

Table 5 shows the results with the unsupervised machine learning classification.

We obtained the best precision result by using the MakeDensityBasedClusterer algorithm (i.e., 0.954) and the best recall is obtained by using the same classification algorithm (0.955) in the PA category feature classification. We

Table 5. Classification results: Precision, Recall, F-Measure and RocArea for classifying the feature categories, computed with six different unsupervised classification algorithms.

Category	Algorithm	Precision	Recall	F-Measure	Roc Area	Time
PI	SimpleKMeans	0.93	0.93	0.93	0.843	1.79
	FarthestFirst	0.92	0.93	0.92	0.839	1.98
	FilteredClusterer	0.91	0.92	0.907	0.841	1.89
	MakeDensityBasedClusterer	0.905	0.918	0.911	0.840	49.06
	Cobweb	0.927	0.919	0.922	0.839	39.58
	DBScan	0.934	0.948	0971	0.836	42.73
PA	SimpleKMeans	0.948	0.927	0.964	0.871	2.37
	FarthestFirst	0.939	0.935	0.936	0.872	3.84
	FilteredClusterer	0.926	0.934	0.92	0.868	3.94
	MakeDensityBasedClusterer	0.954	0.955	0.953	0.845	47.85
	Cobweb	0.926	0.918	0.921	0.862	44.25
	DBScan	0.921	0.924	0.922	0.873	40.68
GA	SimpleKMeans	0.839	0.874	0.856	0.722	1.92
	FarthestFirst	0.801	0.827	0.813	0.747	3.29
	FilteredClusterer	0.833	0.879	0.855	0.779	3.98
	MakeDensityBasedClusterer	0.838	0.875	0.856	0.739	31.48
	Cobweb	0.811	0.825	0.817	0.749	32.19
	DBScan	0.814	0.830	0.821	0.751	35.28
SID	SimpleKMeans	0.827	0.841	0.833	0.719	2.01
	FarthestFirst	0.787	0.798	0.792	0.702	1.93
	FilteredClusterer	0.799	0.778	0.788	0.705	2.16
	MakeDensityBasedClusterer	0.799	0.801	0.799	0.718	37.98
	Cobweb	0.809	0.817	0.812	0.702	41.85
	DBScan	0.789	0.799	0.799	0.793	39.84
NM	SimpleKMeans	0.902	0.919	0.914	0.819	3.06
	FarthestFirst	0.899	0.857	0.877	0.709	2.87
	FilteredClusterer	0.899	0.854	0.876	0.714	2.48
	MakeDensityBasedClusterer	0.923	0.928	0.923	0.858	42.42
	Cobweb	0.876	0.890	0.882	0.744	43.87
	DBScan	0.839	0.875	0.853	0.732	44.28

highlight in addition that with regards to the PA category using the other algorithms the worst precision and recall values are obtained by the DBScan algorithm (0.921 of precision and 0.924 of recall). With the classification of the PI features category, we obtain a precision ranging from 0.91 (with the Filtered-Clusterer algorithm) to 0.934 (with the DBScan algorithm) and a recall ranging from 0.918 (using the MakeDensityBasedClusterer) to 0.948 with the DBSCan algorithm. For the others features categories we obtained the following results:

– with regards to the GA category, a precision ranging from 0.801 (with the FarthestFirst algorithm) to 0.839 (with the SimpleKMeans algorithm) and a recall ranging to 0.825 (with the Coweb algorithm) to 0.879 (with the FilteredClusterer algorithm);
– with regards to the SID category, a precision ranging from 0.787 (with the FarthestFirst algorithm) to 0.809 (with the Coweb algorithm) and a recall ranging to 0.778 (with the FilteredClusterer algorithm) to 0.841 (with the SimpleKMeans algorithm);
– with regards to the NM category, a precision ranging from 0.839 (with the DBScan algorithm) to 0.923 (with the MakeDensityBasedClusterer algorithm) and a recall ranging to 0.854 (with the FilteredClusterer algorithm) to 0.928 (with the MakeDensityBasedClusterer algorithm).

Looking to the tables, the best feature categories, in both the supervised and the unsupervised classifications, are the PA and PI category; for this reason, we perform a feature selection on these categories in order to investigate whether using a small feature set we are able to obtain better results. As matter of fact, the feature selection is employed to improve the ability of classifier in discriminating human and game bot instances and decrease training time. The used feature selection algorithm is the BestFirst, that implements a best-first search strategy to navigate attribute subsets which basically explores a graph by expanding the most promising node chosen according to a specified rule. We obtained that the most discriminating feature in the PI category is just the play time (PI_2), while in PA category there are four best features: sitting (PA_1), earning experience points (PA_2), obtaining items (PA_3) and earning player kill points (PA_5). In order to evaluate if the new features set belonging to PI and PA categories are able to overcome the results obtained in Table 4, we build different (supervised and unsupervised) classifiers. Table 6 shows the results we obtained using the supervised classification algorithms, while Table 7 shows the results we obtained using the unsupervised classification algorithms.

The values of precision and recall are increased for both the features categories we considered. For the PI category, the best precision value (RandomForest) is incremented from 0.952 to 0.954 and the recall one is incremented from 0.953 to 0.984. For the PA category, the best precision value (RandomForest) it is incremented from 0.954 to 0.960 and the recall one is incremented from 0.955 to 0.986. The time analysis confirms the effectiveness of the feature selection step in order to quickly identify the game bot: as matter of fact with the exception of the RandomForest algorithm, the remaining ones are able to learn the classifiers in less than 1 s.

Confirming the trend related to the supervised classification algorithms, also considering the unsupervised classification algorithms, we obtain an increment of the performance. As a matter of fact, the value of precision and recall are increased for both the features categories we considered in the unsupervised machine learning:

Table 6. Feature Selection Classification results: Precision, Recall, F-Measure and RocArea for classifying the feature resulting of the feature selection process with the features of PI and PA categories, computed with six different classification algorithms [44].

Category	Algorithm	Precision	Recall	F-Measure	Roc Area	Time
PI_selected	J48	0.954	0.984	0.969	0.824	0.36
	DecisionStump	0.954	0.984	0.969	0.823	0.05
	HoeffdingTree	0.953	0.985	0.968	0.845	0.41
	RandomForest	0.943	0.944	0.943	0.831	15.06
	RandomTree	0.943	0.944	0.943	0.774	0.37
	REPTree	0.954	0.984	0.969	0.837	0.33
PA_selected	J48	0.958	0.986	0.972	0.870	0.48
	DecisionStump	0.957	0.971	0.964	0.830	0.15
	HoeffdingTree	0.957	0.985	0.971	0.882	0.25
	RandomForest	0.960	0.986	0.973	0.890	56.67
	RandomTree	0.954	0.954	0.954	0.818	0.98
	REPTree	0.959	0.985	0.972	0.889	0.74

Table 7. Feature Selection Classification results: Precision, Recall, F-Measure and RocArea for classifying the feature resulting of the feature selection process with the features of PI and PA categories, computed with six different unsupervised classification algorithms.

Category	Algorithm	Precision	Recall	F-Measure	Roc Area	Time
PI_selected	SimpleKMeans	0.97	0.95	0.95	0.896	0.08
	FarthestFirst	0.99	0.87	0.92	0.867	0.12
	FilteredClusterer	0.97	0.95	0.95	0.891	0.14
	MakeDensityBasedClusterer	0.97	0.95	0.95	0.891	12.04
	Cobweb	0.98	0.84	0.93	0.864	11.27
	DBScan	0.97	0.86	0.93	0.859	12.58
PA_selected	SimpleKMeans	0.96	0.94	0.94	0.848	0.12
	FarthestFirst	0.96	0.94	0.94	0.838	0.24
	FilteredClusterer	0.94	0.92	0.91	0.831	0.18
	MakeDensityBasedClusterer	0.97	0.95	0.95	0.891	12.04
	Cobweb	0.97	0.83	0.88	0.827	14.86
	DBScan	0.96	0.85	0.09	0.839	15.02

– relating to the PI category, the best precision value it is incremented from 0.934 to 0.95;
– relating to the PA category, the best precision value it is incremented from 0.954 to 0.97.

With regards to the time performance analysis, we confirm an increasing number in the time to learn the several unsupervised classifiers.

6 Conclusions

Online games are a widespread form of entertainment allowing users geographically distributed to play together, obtain upgrades for their characters by defeating monsters and enemies, and earn the game currency that can be changed in the real money. In this scenario, same attachers can use game bots to acquire a higher number of game points since game bots are able to play without any interruption disturbing the game of the human players. In this paper the method introduced in [10, 11] is described and largely evaluated. The aims is to discriminate an human user from a game bot classifying a set of behavioral features. We consider a set of features related to the player and to the game. On this features a classification is performed by obtaining in the best case a precision equal to 0.96 and a recall equal to 0.986. Moreover the same evaluation is also repeated by using cluster analysis algorithms and the best obtained results are good (0.99 for the best precision and 0.95 for the best recall). As future work we plan to investigate whether the feature vector we considered in this work is able to detect game bots in social network. Furthermore, we will consider the adoption of Process Mining techniques [45] and Formal Methods [46] in order to extract the game bots patterns with the aim to verify whether are different from the human users one.

Acknowledgements. This work has been partially supported by H2020 EU-funded projects NeCS and C3ISP and EIT-Digital Project HII.

References

1. Adams, E.: Fundamentals of Game Design (2014)
2. Quandt, T., Kröger, S.: Multiplayer: The Social Aspects of Digital Gaming, vol. 3. Routledge, Abingdon (2013)
3. Seay, A.F., Jerome, W.J., Lee, K.S., Kraut, R.E.: Project massive: a study of online gaming communities. In: CHI 2004 Extended Abstracts on Human Factors in Computing Systems, pp. 1421–1424. ACM (2004)
4. Yee, N.: Maps of digital desires: exploring the topography of gender and play in online games. In: Beyond Barbie and Mortal Kombat: New Perspectives on Gender and Gaming, pp. 83–89 (2008)
5. Griffiths, M.D., Davies, M.N., Chappell, D.: Online computer gaming: a comparison of adolescent and adult gamers. J. Adolesc. **27**, 87–96 (2004)
6. Chen, Y.C., Chen, P.S., Song, R., Korba, L.: Online gaming crime and security issue-cases and countermeasures from Taiwan. In: PST, pp. 131–136 (2004)
7. Paulson, R.A., Weber, J.E.: Cyberextortion: an overview of distributed denial of service attacks against online gaming companies. Issues Inf. Syst. **7**, 52–56 (2006)
8. Yampolskiy, R.V., Govindaraju, V.: Embedded noninteractive continuous bot detection. Comput. Entertain. (CIE) **5**, 7 (2008)
9. Kang, A.R., Jeong, S.H., Mohaisen, A., Kim, H.K.: Multimodal game bot detection using user behavioral characteristics. SpringerPlus **5**, 523 (2016)
10. Cabello, E., Cardoso, J., Ludwig, A., Maciaszek, L.A., van Sinderen, M. (eds.): ICSOFT 2016. CCIS, vol. 743. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62569-0

11. Bernardi, M.L., Cimitile, M., Mercaldo, F.: A time series classification approach to game bot detection. In: Proceeding of the 7th ACM International Conference on Web Intelligence, Mining and Semantics, pp. 512–519 (2017)
12. Varvello, M., Voelker, G.M.: Second life: a social network of humans and bots. In: Proceedings of the 20th International Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV 2010, pp. 9–14. ACM, New York (2010)
13. Oh, J., Borbora, Z.H., Sharma, D., Srivastava, J.: Bot detection based on social interactions in MMORPGs. In: 2013 International Conference on Social Computing (SocialCom), pp. 536–543. IEEE (2013)
14. Kang, A.R., Woo, J., Park, J., Kim, H.K.: Online game bot detection based on party-play log analysis. Comput. Math. Appl. **65**, 1384–1395 (2013)
15. Kim, H., Hong, S., Kim, J.: Detection of auto programs for MMORPGs. In: Zhang, S., Jarvis, R. (eds.) AI 2005. LNCS (LNAI), vol. 3809, pp. 1281–1284. Springer, Heidelberg (2005). https://doi.org/10.1007/11589990_187
16. Chen, K.T., Pao, H.K.K., Chang, H.C.: Game bot identification based on manifold learning. In: Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games, pp. 21–26. ACM (2008)
17. Chen, K.-T., Liao, A., Pao, H.-K.K., Chu, H.-H.: Game bot detection based on avatar trajectory. In: Stevens, S.M., Saldamarco, S.J. (eds.) ICEC 2008. LNCS, vol. 5309, pp. 94–105. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89222-9_11
18. Chen, K.T., Jiang, J.W., Huang, P., Chu, H.H., Lei, C.L., Chen, W.C.: Identifying MMORPG bots: a traffic analysis approach. In: Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, ACE 2006. ACM, New York (2006)
19. Kwon, H., Mohaisen, A., Woo, J., Kim, Y., Lee, E., Kim, H.K.: Crime scene reconstruction: online gold farming network analysis. IEEE Trans. Inf. Forensics Secur. **12**, 544–556 (2017)
20. Kim, H., Yang, S., Kim, H.K.: Crime scene re-investigation: a postmortem analysis of game account stealers' behaviors. CoRR abs/1705.00242 (2017)
21. Thawonmas, R., Kashifuji, Y., Chen, K.T.: Detection of MMORPG bots based on behavior analysis. In: Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology, pp. 91–94. ACM (2008)
22. Kashifuji, Y.: Detection of MMORPG bots based on behavior analysis. In: ACE 2008 (2008)
23. Hilaire, S., Kim, H., Kim, C.: How to deal with bot scum in MMORPGs? In: 2010 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR), pp. 1–6. IEEE (2010)
24. Mishima, Y., Fukuda, K., Esaki, H.: An analysis of players and bots behaviors in MMORPG. In: 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), pp. 870–876. IEEE (2013)
25. Chung, Y., Park, C.Y., Kim, N., Cho, H., Yoon, T.B., Lee, H., Lee, J.: A behavior analysis-based game bot detection approach considering various play styles. CoRR abs/1509.02458 (2015)
26. Mitchell, T.M.: Machine learning and data mining. Commun. ACM **42**, 30–36 (1999)

27. Michalski, R.S., Carbonell, J.G., Mitchell, T.M.: Machine Learning: An Artificial Intelligence Approach. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-662-12405-5
28. Estai, M., Kanagasingam, Y., Xiao, D., Vignarajan, J., Bunt, S., Kruger, E., Tennant, M.: End-user acceptance of a cloud-based teledentistry system and Android phone app for remote screening for oral diseases. J. Telemed. Telecare **23**, 44–52 (2017)
29. Canfora, G., De Lorenzo, A., Medvet, E., Mercaldo, F., Visaggio, C.A.: Effectiveness of opcode ngrams for detection of multi family Android malware. In: 2015 10th International Conference on Availability, Reliability and Security (ARES), pp. 333–340. IEEE (2015)
30. Canfora, G., Mercaldo, F., Visaggio, C.A.: A classifier of malicious Android applications. In: 2013 Eighth International Conference on Availability, Reliability and Security (ARES), pp. 607–614. IEEE (2013)
31. Ling, C.X., Yang, Q., Wang, J., Zhang, S.: Decision trees with minimal costs. In: Proceedings of the Twenty-First International Conference on Machine learning, p. 69. ACM (2004)
32. Jin, C., De-Lin, L., Fen-Xiang, M.: An improved ID3 decision tree algorithm. In: 4th International Conference on Computer Science and Education, ICCSE 2009, pp. 127–130. IEEE (2009)
33. Pang, J., Huang, Q., Jiang, S.: Multiple instance boost using graph embedding based decision stump for pedestrian detection. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008. LNCS, vol. 5305, pp. 541–552. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88693-8_40
34. Hang, Y., Fong, S.: Investigating the impact of bursty traffic on Hoeffding Tree Algorithm in stream mining over internet. In: 2010 Second International Conference on Evolving Internet (INTERNET), pp. 147–152. IEEE (2010)
35. Liaw, A., Wiener, M., et al.: Classification and regression by randomForest. R News **2**, 18–22 (2002)
36. Cutler, A., Zhao, G.: Pert-perfect random tree ensembles. Comput. Sci. Stat. **33**, 490–497 (2001)
37. Zhao, Y., Zhang, Y.: Comparison of decision tree methods for finding active objects. Adv. Space Res. **41**, 1955–1959 (2008)
38. Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C., Silverman, R., Wu, A.Y.: The analysis of a simple k-means clustering algorithm. In: Proceedings of the Sixteenth Annual Symposium on Computational Geometry, pp. 100–109. ACM (2000)
39. Kumar, M., et al.: An optimized farthest first clustering algorithm. In: 2013 Nirma University International Conference on Engineering (NUiCONE), pp. 1–5. IEEE (2013)
40. Panda, M., Patra, M.: A novel classification via clustering method for anomaly based network intrusion detection system. Int. J. Recent Trends Eng. **2**, 1–6 (2009)
41. Pandey, A.K., Pandey, P., Jaiswal, K., Sen, A.K.: Datamining clustering techniques in the prediction of heart disease using attribute selection method. Heart Dis. **14**, 16–17 (2013)
42. Fisher, D.H.: Knowledge acquisition via incremental conceptual clustering. Mach. Learn. **2**, 139–172 (1987)
43. Dua, S., Du, X.: Data Mining and Machine Learning in Cybersecurity. CRC Press, Boca Raton (2016)

44. Bernardi, M.L., Cimitile, M., Distante, D., Mercaldo, F.: Game bot detection in online role player game through behavioural features. In: Proceeding of the 12th International Conference on Software Technologies (2017)
45. Bernardi, M.L., Cimitile, M., Di Francescomarino, C., Maggi, F.M.: Do activity lifecycles affect the validity of a business rule in a business process? Inf. Syst. **62**, 42–59 (2016)
46. Francesco, N.D., Lettieri, G., Santone, A., Vaglini, G.: Heuristic search for equivalence checking. Softw. Syst. Model. **15**, 513–530 (2016)

Genrih, a Runtime State Analysis System for Deciding the Applicability of Dynamic Software Updates

Oleg Šelajev[1](✉) and Allan Raundahl Gregersen[2](✉)

[1] University of Tartu, Tartu, Estonia
shelajev@gmail.com
[2] ZeroTurnaround, Tartu, Estonia
allan.gregersen@zeroturnaround.com

Abstract. Dynamic Software updating (DSU) systems enable applications to be upgraded without service interruption. However, the implications of changed program assumptions may result in unwanted runtime phenomena after the dynamic update if the momentary state of the application does not satisfy those changed assumptions. Hence, in order to enable dynamic updates in a safe manner, the updating mechanism needs to reason about the runtime state at update time.

We present a runtime state analysis system, Genrih, that enhances an existing dynamic update system with the ability to take automated informed decisions concerning the safety of a particular program update. Genrih will determine if the automated default state transformations of the underlying DSU system are sufficient for the given update. In Genrih the atomic changes that constitute the update patch are analyzed in combination with the present runtime state of the application. Based on that analysis Genrih determines whether updating the system will lead to observable unwanted runtime phenomena.

While Genrih is powerful enough to block updates until the runtime state satisfies the update to allow for a safe update, for practical purposes it observes the runtime state and produces notifications for enhanced analysis and crash management. The practical evaluation shows that the designed system imposes acceptable overhead and can help educate developers about runtime phenomena.

Keywords: Dynamic software update · Runtime phenomena
State analysis · Reliability · Availability

1 Introduction

Software projects for existing applications inevitable grow and evolve continuously. Efficiently and safely accommodating changes to the applications is as relevant as ever. The Dynamic Software Update (DSU) research field deals with the ability to apply changes to a system at runtime without service interruption

© Springer International Publishing AG, part of Springer Nature 2018
E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 135–159, 2018.
https://doi.org/10.1007/978-3-319-93641-3_7

and without affecting the behavior of the application. The historical motivation for contributing to the DSU field has its origin in the desire to generically handle runtime updates of mission-critical applications that cannot tolerate maintenance downtime. In the past decades several strong contributions have been made in this field, [1–5,11]. While the ability to update mission-critical systems is instrumental to achieve the ultimate goal of DSU, research on the safety, [6–8] and in particular the finding that to achieve guaranteed safety, a portion of human intervention and input is required to apply safe dynamic updates, [5,9].

Applying updates while applications run is a complex task which roughly divides into two main parts; (1) ensuring that the new behavior of the updated program is reflected after having applied the update and (2) transforming the existing runtime state of the application to accommodate the needs of the new program specification.

The first part has been addressed in the literature by making the programming language runtime aware of possible dynamic updates, [10,11] employing architectures that make dynamic updates easier, [2]. An alternative approach has been to extend and enhance existing runtime platforms dynamically to add support for dynamic updates, [4,12].

The second part constitutes a more complex issue, which in the general case of guaranteeing the safety of a particular dynamic update, has been shown to be unsolvable, [1]. Hence, the human intervention required from program experts to transform the runtime state to conform to the structure of the new version is tedious. The current state-of-the-art way to handle necessary change to the runtime state at the update time is to manually specify state transformation functions that will convert the existing runtime state into a representation suitable for the new version of the program.

Since the safety of mission-critical systems is paramount, and we do not know how to automate the state transformation fully, the DSU systems cannot risk employing automatic state transfer solutions, [8]. However, defining these state transformation functions is not easy, time-consuming, and error-prone. Moreover, requiring the manual intervention of the programmer is not always appropriate. For example, applying dynamic updates in the development environment to avoid long pauses.

The current state of the art DSU systems that do not require manual actions from the developers, and hence cannot fully guarantee safety, statically analyze the code to determine when it can be updated, [13].

With the above generic issues in mind, this paper suggests a solution, Genrih that grants developers immediate feedback on the update safety for specific program patches based on the program runtime state at the time of the dynamic update attempt. We limit ourselves to investigating DSU of Java applications. Java represents a popular member of the family of the statically typed object-oriented programming languages and has multiple DSU solutions available.

In Genrih, developers need not specify safepoints at which the DSU system can perform the update in a safe manner. Genrih, will, depending on the configuration, either block unsafe updates until the program reaches a safe runtime state (possibly indefinitely) or allow unsafe updates to be carried out while providing detailed logging information about particular known runtime phenomena, [14]. While applying unsafe updates sounds like a bad idea, the main reasoning behind this operational mode, is that Genrih can be utilized as an educational tool for developers working with a state-of-the-art DSU system like JRebel, [4].

Having obtained feedback from many clients of JRebel, one of the main difficulties when working with a DSU at development time, is that whenever an app failure happens one of the first things developers ask themselves is whether the failure was due to the DSU. Genrih greatly improves this scenario, in the sense that it provides immediate feedback on possible unwanted specific side-effects that can be expected from the updates. This is a strong step towards more dynamic update safety since developers will learn how to best architect their systems to ease possible dynamic software updates.

The main contribution of this paper is to propose a system that automatically decides at runtime if the automatic state mapping of a given update mechanism is sufficient for the update. The proposed system is orthogonal to the DSU solution used for the updates. The practical evaluation is performed on a prototype which is built with JRebel and Rubah updating functionality in mind.

This paper is an extended version of the 2017 ICSOFT paper "Using Runtime State Analysis to Decide Applicability of Dynamic Software Updates", [15]. While some parts of this paper have been completely rewritten, e.g., abstract, introduction and conclusion, others have been improved and sharpened. Moreover, in this paper we've added a second DSU system, Rubah, to underpin the generality of our approach, Genrih, and performed additional tests and performance evaluation of Genrih's approach.

The remainder of this paper is as follows. In Sect. 2 we establish the framework for the terminology used in the paper as well, as well as explaining the basic idea behind Genrih. Section 3 will then turn to how we implemented Genrih by going through the main components of the system. This ends up with a prototype implementation that will be utilized is experiments as explained in Sect. 4. We evaluate the performance implications in Sect. 5 and discuss our main findings of the research in Sect. 6. Section 7 details related work while we finally conclude in Sect. 8.

2 Background

Looking at the full set of possible code changes that can be applied to Java application classes, a subset can potentially lead to certain runtime phenomena if applied without concern, [16]. However, if the DSU system applies changes deterministically, the manifestation of those runtime phenomena after applying a code change between the old and the new class is determined solely by the DSU and the runtime state at the exact time of the update. Hence, every change can potentially cause only a limited number of runtime phenomena. One example would be when a new field is added to a class which by nature will leave the field uninitialized. Logically, this can cause a *NullPointerException* in the event the field is being read from an instance that existed before the update. This is an example of the so-called absent application state phenomena. More interestingly, adding a new field cannot lead to other types of runtime phenomena, like for example "phantom objects" phenomena or any other from the list below. In the paper describing runtime phenomena, [14], Gregersen and Jørgensen have mapped certain observed phenomena to the individual class changes that were responsible for them.

The following paragraphs list the set of possible Java code changes and their related runtime phenomena that will be used throughout this paper. The definitions of the runtime phenomena are taken from the work conducted by Gregersen and Jørgensen, [14].

Phantom objects are live objects whose classes have been removed or invalidated by a dynamic update, [14]. Changes that can introduce phantom objects are, for example, removing a class, adding modifier abstract to the class definition, replacing a class with an interface. Indeed, if the update adds a modifier abstract to a definition of a class or replaces it with the interface, it means that in the new version of the application, no instances of that class can be instantiated. Hence, in the new version of the application, the very existence of such objects should by definition be impossible. Any instances of this class that did exist before the update will turn into phantom objects that have no place in the correctly behaving Java application.

Absent state is defined as the situation in which objects or classes having been created in a previous version once migrated to the new version,lacks a portion of the expected state, [14]. Adding an instance or static field to a class, which might not be initialized during the update, as previously described is probably the most straightforward example of introducing the absent state phenomena. There are a number of individual code changes that can lead to the manifestation of the absent state phenomena. For example, adding a new subclass with the intent to differentiate between objects using polymorphism can cause it. Since no objects existing prior the update can be of the new subclass, differentiating by type will fail despite the fact that it might succeed during a fresh run, where objects are created based upon the new assumptions there were introduced with the new design. Another example of a specific code change that can cause the absent state phenomena is if the declared superclass (extends clause) of a class is changed. All existing objects of the class are not reconstructed, so the set of fields declared within the new super hierarchy will not have been initialized since the constructor for those objects ran prior to the dynamic update. Yet another example would be to remove the static modifier from an inner class. Behind the scenes of the Java compilation phase (javac) what happens is that a new synthetic field for referencing the outer instance from within the code of the inner class is generated. It will not be possible to automatically deduce the outer instance during the update so the value of this new synthetic field will stay absent.

Lost state takes place when existing state that can be used to differentiate between objects is lost. Removing fields or changing the type of a field can produce the lost state phenomena. In these cases, the information held by the field prior to the update is inaccessible from the new code by virtue of the state being either removed or overwritten with a default value of the new type, possibly even null.

Oblivious update phenomenon is the absence of an expected runtime effect that would have occurred if the system was started from scratch. For example, a change to a constructor does not affect all instances having been previously constructor before the update, [14]. Like constructor changes the class-level variant of constructors, namely static initializers, can cause oblivious update phenomena when changed since a class can be loaded only once within the JVM.

There are a few other runtime phenomena that are observable after updates, but in this paper we focus on these four key phenomena being described and analyzed in the prior work by Gregersen and Jørgensen, [14]. The description of the runtime phenomena given above maps the runtime phenomena to individual changes that cause them. Naturally, if we reverse the mapping, then we can map individual code changes to potential application level runtime phenomena. With this at hand, knowing the set of individual code changes that constitute a dynamic update, it will be possible to predict potential unwanted phenomena if applied by a given DSU solution.

The phenomena listed in the Gregersen's work is orthogonal to the exact approach used by the dynamic update system and are solely dependent on the runtime state of the application. These phenomena occur only when the application state satisfies certain conditions, which can be expressed as queries to the runtime state that evaluate if phenomena dictated by the changes might be observed. The non-determinism of the phenomena and their dependency of the runtime state is analyzed in the same work which introduced them, [14].

As an example of a query determining if the runtime phenomena may occur on any access to the instances of a given class initialized before the update, would be if the answer is yes to the question "are instances of the class or any of its subclasses present in the application?" given the runtime state at the time of the update.

The above simple query can be directly applied in a more useful scenario, and it can determine if the following runtime phenomena might be observed: phantom objects due to a class removed code change, adding abstract modifier code change or a changing a class to an interface code change. The base query for determining lost state phenomenon would also be: "are instances of the class or any of its subclasses present in the application?", however, for this case it can be specialized to analyze if the field of those object instances can be used to differentiate them.

Oblivious update caused by changing the constructor code can be predicted by the same query, however, if it is caused by changing a static initializer of a class, then a simpler query whether the class is loaded into the JVM suffices since static initializers are not executed for every object instance as mentioned previously.

By describing the queries to determine runtime phenomena, we implicitly presented a mapping between the runtime phenomena and the changes that cause them. The reverse mapping of possible runtime phenomena caused by the changes to the classes of the applications can serve as a good first approximation for the mapping database approach for verifying the update safety. Further work can improve on those mappings and include more changes to effects mappings to cover more ground. The exact implementation of the mapping that was used in the evaluation of the approach is discussed further in the following section.

Note, that it is possible to derive less conservative queries for specific runtime phenomena, but for this paper, we used the most straightforward ones, usually saying that the phenomenon is possible to occur if the instances of the changed class exist on the heap. We deliberately kept it simple, since, for the sake of this work, we wanted to emphasize the fact that with the combination of easily available data of individual code changes and simple queries on the runtime state, we're able to build a system that is useful in practice. Obviously, static analysis of, e.g., call sites of new fields could be used to rule out the possibility for certain phenomena, thus improving the precision of the approach, but not the basic idea behind it.

The approach described in the Static Analysis for Dynamic Updates tool paper, [17], is capable of discovering the atomic changes that were found to be responsible for the above-mentioned phenomena during the experiment. The tool compares application classes one by one and returns the list of individual changes in them. In this paper, we present a runtime state analysis engine that provides an insight into the runtime state of the application and is capable of serving the queries mentioned above. This runtime state analysis engine forms the last component of the system required to establish if an update can lead to the observable runtime phenomena. Thus, the main components of Genrih are (1) the class diff tool as described above, (2) a database to map these changes to the corresponding runtime phenomena, (3) an analytic engine to inspect the runtime state can provably establish if the update is safe from introducing these unwanted effects into the updated application and (4) the DSU system of choice to perform the actual dynamic updates.

3 A System for Predicting Runtime Phenomena

Deciding if the state transformation functions are adequate for a given update is equivalent to analyzing whether the update will produce runtime phenomena. The system implementing the dynamic state analysis to predict if a given update can result in observable runtime phenomena is orthogonal to the actual DSU solution performing the update. The architecture of the integration with DSU solutions is quite straightforward. It requires the access to the original and changed versions of the classes involved in the update before the actual update happening; then it will run the analysis of the application state on the heap and determine if applying the update is safe or if any application level phenomena can be observed afterward.

The system to predict the potential faultiness of a dynamic update consists of the following components: an runtime state analysis engine that is able

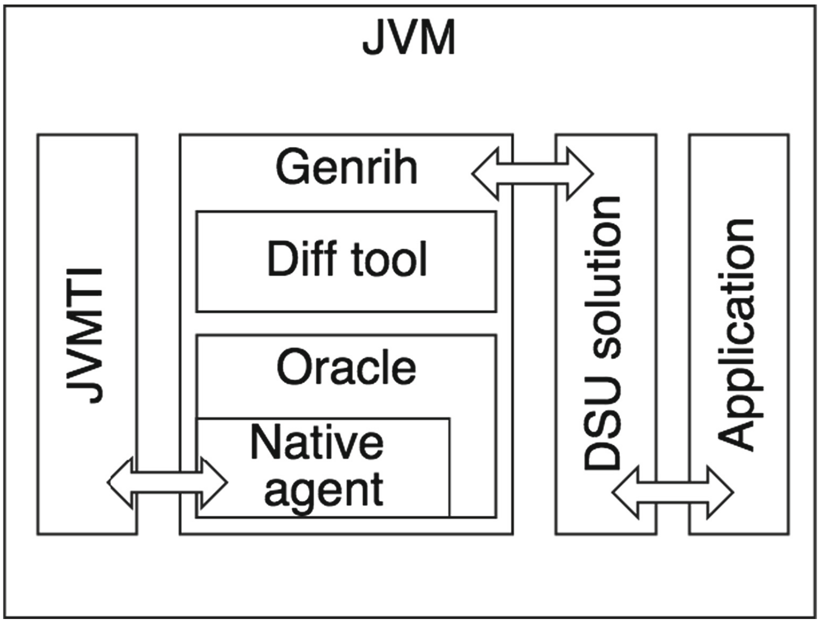


Fig. 1. Architechture of runtime analysis system.

to respond to queries about the current runtime state; a class diff tool that determines which changes has occurred between two versions of the application classes; a mapping between the changes into the possible runtime phenomena caused by these changes, and a world stopper that can pause the JVM to ensure that the analysis is sound, and the application state does not change between the analysis and the execution of the update by the DSU solution.

The overall architecture of the designed system is depicted by Fig. 1, originally produced for the ICSOFT version of this paper, [15].

Genrih is developed as a standalone component separate from both application code and the DSU solution that performs updates. It embeds both the diff tool to compare class files and the runtime state analysis engine, which uses JVM native agent capabilities to inspect the runtime state. The arrows on the Fig. 1, show the main communication patterns between the components. Genrih does not act upon the application itself, but rather integrates with the DSU solution to obtain changed classes and then it notifies about the update safety. The native agent inside the system uses the Java Virtual Machine Tool Interface (JVMTI), [18] to gain access to the runtime state that is not otherwise possible to obtain from regular Java code. Note that the capabilities of non-native *javaagent* or application level code are not enough to perform the analysis of the Java heap to the extent necessary.

The rest of this section describes responsibilities and a possible implementation of each component and how they all come together to predict the faultiness.

3.1 State Analysis Engine

The JVM stores objects on the heap. The memory on the heap is managed: the memory for new objects is allocated automatically, reachable objects are often moved around, unreachable objects are garbage collected and the memory claimed by them can be reused.

As we saw above most of the queries to determine the possibility of a phenomena consist of two questions: (1) is class T loaded into the JVM and (2) are there initialized and not yet garbage collected instances of the class T, sometimes including its subclasses.

Luckily, we can answer both of these questions for a given class T by leveraging the JVM tool interface (JVMTI) and a native agent. JVMTI provides a way to inspect the state of the running JVM program and influence its execution. Determining if a class is loaded in the JVM can be done from the JVM itself, especially if we are not interested in the class loading details and can query just the system class loader. We can use the *findLoadedClass* method of the ClassLoader class to obtain this information.

To count the number of the instances of a class T, we can make use of JVMTI's "iterate through heap" function that takes a class which instances to iterate and a callback function to call for every instance. For our purposes, the callback function just increments a counter for every found instance. If after the invocation of the "iterate through heap" with the given class argument the result is non-zero, there are instances of the given class.

The native agent approach to leverage iterating through heap is general enough to cover more complex queries that might be necessary to introduce for possible future enhancements of the system. For example, if one needs to investigate a removal of the field and the possibility of lost state phenomenon, a more complex callback that inspects every instance found by the iterate through the heap function to determine if the field in question has a distinguishing value or not. If all the objects have the field initialized to the default value for the field type, removing the field does not lose any data.

The discussion section contains more details about which queries were implemented in the prototype of the described system and why.

3.2 Class Diff Tool

The DSU solution must have access to the new versions of the classes to update the definitions in the JVM. We can use a simple cache mechanism to store the bytecode of the loaded classes by the class name. To reduce the memory footprint, we can potentially ask the DSU solution if the class in question is reloadable, meaning if there is a possibility that the bytecode for that will change and avoid storing the bytecode for classes that won't be updated. Alternatively, caching to disk can be utilized, but such a scheme can introduce additional performance overhead. However, this becomes an engineering challenge, so for the purposes of this paper, we implemented the in-memory approach.

The class diff tool, [17], consumes two sets of classes: old and new; and produces a list of *events* that describe how the classes in the new set differ from their respective counterparts in the old set.

For the class definitions that are different in the old version and the new version, there are three main results of the diff analysis: old class does not exist, new class does not exist, and both classes exist. First two cases are naturally mapped to the *class_added* and *class_removed* changes respectively. If both class definitions exist, class diff tool analyzes them for the differences. The class diff tool described in the Static Analysis for Dynamic Updates paper, [17] does exactly that, it takes two definitions of a class and returns a list of the events: *new_instance_field_added*, *new_static_method* event, etc. Each individual change event has a reference to the class and the class member: method, field, or initializer that were affected by the change.

The analysis engine can run the diff tool on every class for which both the old and the new versions are present and obtain the list of exact changes between them. Then the entries within the list are mapped to the possible runtime phenomena they might surface which in turn is fed to the runtime state analysis component to determine the possibility of the phenomena given the current runtime state of the application.

3.3 Changes to Phenomena to Runtime State Queries

We need to map the exact changes that occurred between the old and the new versions of the classes involved in the dynamic update. During an update, we have the class definitions for both old and new versions of the application. The output of the class diff tool described above is the list of change events that are found in these versions.

In the current work, we investigate the following changes that were previously found to be causing the runtime phenomena, [14]. The first column of the Table 1, taken from the original version of this paper, [15], specifies the changes, the second lists the runtime phenomena they can produce, and the third column describes the queries, which reveal whether the phenomena might be observable.

From all of the changes that can lead to runtime phenomena which at the same time are recognized by the system we designed, we handpicked a subset for showcasing the approach. While other changes are also interesting, these have been previously recognized to lead to the runtime phenomena described above, [14].

In future research, the mapping for the changes can be more thorough, however, being able to predict the faultiness with respect to the runtime phenomena observation after the dynamic update containing these changes is a useful result in itself.

The mapping provided above can be directly translated to code via a series of if-else statements, where the class T is the class currently being diffed. When the queries that correspond to the possibility of observing the runtime phenomena are obtained, the runtime state analysis engine evaluates them using the queries it knows how to answer. The result shows whether the update is safe from the application level runtime phenomena, with respect to the phenomena and the changes that we analyze.

Table 1. Changes to phenomena to queries table.

Change type	Phenomena	Analysis queries
Class T made abstract change	Phantom objects: instances of class T are on the heap	Class T is loaded and instances of T are on the heap
Class T removed change	Phantom objects: instances of class T or subclasses are on the heap	Class T is loaded and instances of T or subclasses are on the heap
Constructor of class T changed	Oblivious update: existing instances have run previous version of constructor	Class T is loaded and instances of T or subclasses are on the heap
New instance field change in class T	Absent state: old instances of T won't have the new field initialized	Class T is loaded and instances of T or subclasses are on the heap
New static field change in class T	Absent state: static field might not be initialized	Class T is loaded
Static initializer changed in class T	Oblivious update: new version of static initializer is not executed	Class T is loaded
Super class of class T changed	Absent state: instances of T or subclasses might not have field of the new superclass initialized	Class T is loaded and instances of T or subclasses are on the heap or the hierarchy from the new superclass to Object does not declare any fields
Modifier static removed from inner class T	Absent state: implicit field out on instances of T is not initialized	Class T is loaded and instances of T or subclasses are on the heap

3.4 World Stopper

A JVM embodies a multithreaded environment where different threads, like the garbage collector, mutate the global state all the time. The direct consequence of this is that to ensure the soundness of the analysis, we have to synchronize the analysis and the update with the JVM activities external to the DSU solution at hand. One way to do this is to rely on the JVM pausing its work for the internal bookkeeping. However, this might not be utterly portable, so the more direct solution is to use the JVMTI thread suspending functionality and iterate over all threads that are not involved in the dynamic update. Stopping the threads for the analysis bears an obvious performance overhead, which we measure in the practical part of the current research.

On the other hand, stopping the JVM for the analysis adds the benefit of knowing exactly what methods are currently active on the stack. If a method

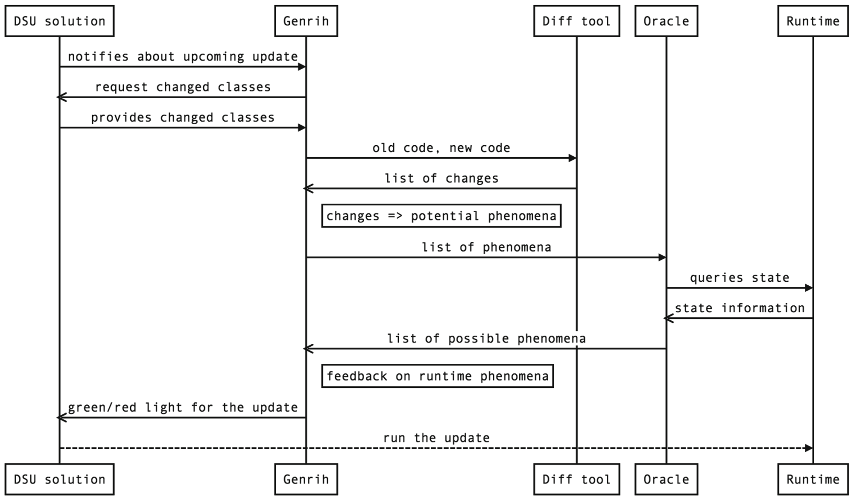


Fig. 2. Update process with Genrih.

body or its signature have been changed in the update and the method is currently running, applying the update can lead to various unwanted side-effects. For example, if the update removed a method which currently executing code tries to call, the best the system can do is to throw a *NoSuchMethodErrors* to communicate inability to locate the method in the updated code. Moreover, stopping the world for the analysis allows the system to inspect if methods that have been changed are currently active. Such runtime check alone can simplify the type safety of the update process by preventing the updates that modify the currently active methods from being applied. However, in the spirit of not intervening with the update process, one can emit a notification that a currently active method is changed and possible side-effects including among others exceptions about class members not found.

In the following subsections, we describe how our system has been integrated into two state-of-the-art DSU systems for Java, namely JRebel and Rubah.

3.5 A Prototype to Enhance JRebel

We implemented a prototype of the system to predict if runtime phenomena may surface when specific updates are applied dynamically. We call the system Genrih and it is integrated with JRebel dynamic updating functionality.

The general workflow of performing a dynamic update with the runtime state analysis is illustrated in the Fig. 2 from the original version of the current paper [15].

The runtime state analysis system marked as Genrih in the Fig. 2 receives a request from the DSU solution that an update is available and the list of classes that are going to be involved in the current update. These classes are diffed to obtain the exact changes in the update. The system then stops all the activity in the JVM using the world stopper described above. At this moment no state can be mutated in the application, so we run the analysis of the heap by evaluating the queries that are mapped to the changes. The dynamic update proceeds as follows, if the update is runtime phenomena free at the current point in time, we signal the DSU solution to continue with the update and replace the class definitions involved. After this process finishes, we can resume the paused threads and report that the update has successfully finished.

Otherwise, Genrih still resumes the threads, but schedule the analysis after a small random delay up to 500 ms hoping that the runtime state of the application will have changed by then, removing the objects that are responsible for the possible phenomena after the update. If the following analysis runs show that the state has not changed enough, and the runtime phenomena are still possible, Genrih emits a notification to the developer saying what runtime phenomena are possible and what change is causing it and if known, which objects contain the state leading to the observable difference in behavior.

In the actual system which enhances the existing development time DSU solutions, after showing the notification, Genrih should allow the system to proceed with the update, not to stall the development process.

The exact details of how the DSU solution we integrate with is performing the update are orthogonal to the safety evaluation. Thus, we can treat it as a black box. The only integration points that we are interested in are:

– a signal that the update is available;
– a list of the classes involved in the update;
– the functionality to signal if Genrih determined whether the update is safe to apply.

These requirements are relatively humble, and the integration with an existing DSU solutions is fairly straightforward.

Genrih implements the runtime state analysis engine, the mapping functionality of class changes to potential side effects, integrations with JRebel and Rubah, and the use of a class diff tool.

3.6 Enhancing Rubah

Rubah is another state-of-the-art dynamic software update system for Java [5]. It was developed in an academic setting and the source code is publicly available [19], which made it an ideal candidate system to integrate with Genrih. In this section, we discuss how the update procedure of Rubah works on top of the stock JVM, how it is different from JRebel's seamless and completely transparent approach, and whether it complicates the runtime state analysis Genrih has to perform.

Rubah implements the dynamic updates entirely through libraries and application bytecode processing, and the main contributions lie within the algorithms relating to post-update state transformations. Rubah performs those algorithms in a lazy and parallel manner, which reduce the time required for the update. Moreover, the steady-state performance overhead imposed by Rubah before and after applying updates is quite moderate.

Rubah is primarily built to apply dynamic updates to applications at production time. Thus it tries to make the update procedure as precise and deterministic as possible, sacrificing the developer transparency. It requires quite a bit of manual manipulation by the developer to retrofit the program to be Rubah compatible. Likewise, Rubah requires human intervention to prepare and trigger an update. Below is a general description of how Rubah operates.

To run a Rubah-enabled application, developers need to start Rubah instead of their own application directly. This allows Rubah to load and manage the application using a combination of a custom class loading and class versioning scheme. Rubah utilizes the class versioning scheme to distinguish between the classes in the old and new versions of the application.

The manual developer intervention required to retrofit the programs to be updatable by Rubah include the following. Rubah offers a thread abstraction 'RubahThread', which is compatible with the JDK thread API and is a drop-in replacement for that. The Rubah runtime has knowledge of RubahThreads and can stop and restart them during the update. If all application threads are Rubah threads, the application can be updated, otherwise, the update results in a crash due to the fact Rubah will try to stop normal Java threads. To ensure that the update is seemingly safe, Rubah does not automatically allow the program to be updated at any moment in time. The developer has to specify markers for safepoints for when the update can be applied. While the marker is a simple call 'Rubah.update("marker");' it does require a deep understanding of the update procedure by the developer. On top of that, the developer has to generate descriptor files for all versions of the application. Such a descriptor file is a list of the classes of the application and some meta-information about them. There is a tool for descriptor generation available for Rubah. Based on the descriptor files and the application jar files for both versions in the update, Rubah generates a stub state migration class. That class has to be available to the update and specifies the logic of transforming the objects from the old representation to the new.

During the update, Rubah performs the following: it loads the new version of the application in memory, stops all running threads in the application, migrates the program state transforming the objects using the generated transformation classes, migrates the control flow of the program by restarting the Rubah threads. After the update, the freshly created Rubah threads run the new version of the program on the transformed runtime state. There are corollaries of the update procedure, which are important for devising a runtime state analysis system and, especially, for applying Rubah for the DSU in development setting this work concentrates on.

The update procedure disregards and loses any state that is held on the stack of the currently running threads. The Rubah threads get restarted, and the normal Java threads just crash, so any information that is stored in the local variables of the method call stack will get lost. Arguably, it is not a problem for the production time dynamic software update, where one would expect the application to be fault-tolerant. The stub state transformations Rubah generates are quite limited in functionality, field values of fields which do not change between the versions are copied into the new objects, but introducing new fields, changing field types or introducing new static fields are handled by just changing the class schema and initializing the fields to the default JVM values: 0, false, null. While in production such conservative approach is beneficial, the developer has the full control of the migration code and assumes no "magic" updates, during development time it limits the amount of update Rubah can safely perform without the runtime phenomena occurring (assuming the developer does not modify the state migration file which is a counter-productive way to spend time).

A default Rubah update that includes the class schema transformations, for example, adding a new instance field, would almost certainly cause a runtime phenomena if the objects transformed reside in the Java memory during the update. Which makes it an excellent setting to test Genrih.

To integrate Genrih with Rubah one needs the same interface to the underlying DSU solution as we described in the previous section about JRebel. It needs to receive a signal about the update happening, the list of changed classes, and a way to emit a notification to the developer if the update will potentially cause runtime phenomena.

Rubah's source code is fairly well organized, the update is handled by a state machine in the 'rubah.runtime.state.States' class. It is a natural entry point to inject Genrih's code into the Rubah runtime.

Rubah threads are the extension of the normal JDK threads, so Genrih's code operating on stopping and resuming threads for the runtime state analysis works without modifications. Heap analysis is also DSU agnostic with the only exception that the conservative queries to the runtime state can lead to more false positives under a more powerful DSU which can handle more state transformations correctly out of the box. Given Rubah's approach to leave all state transformation code on the developer, and reasonable lack of interest to specify migrations for the updates manually in the development time DSU problems, Genrih's conservative approach works well for Rubah.

The next section describes the experiment of updating a real-world game application with the runtime analysis system capable of predicting runtime phenomena occurring because of these changes to the code.

4 Experiment

For JRebel, we have evaluated the designed system on the update scenarios of a Space Invaders game that Gregersen and Jørgensen have used to prove the existence of the runtime phenomena, [14]. The choice of the application for the experiment is influenced by the lack of the systematic benchmarks or analysis of the development time DSU solutions. Also, reusing a code base that certainly contains several versions of the application different enough to produce runtime phenomena after the updates is more relevant to the current work than performing the experiment on an arbitrary code.

The performed experiment was designed to provide information about two hypotheses:

1. The system can predict that an update will not cause side effects or provide an immediately qualified feedback regarding possible runtime phenomena.
2. When or if the runtime state at the update time will not lead to the side effects, the system carries the update out without the considerable overhead.

The experiment process follows the given procedure. Two versions of the Space Invaders game are manually investigated to find out the changes that correspond to the update. Both versions of the game are started to determine what is the expected behavior of the program.

After some state is reached using the old version of the code, we update the code base to the new version and build the application without stopping the use of it. If the update is successful, we try to observe the runtime phenomena predicted by the manual code analysis. If we observe the side effects, we consult the output of Genrih to verify that the phenomena were predicted, and the notification of its effects is present.

Although the experiment procedure is not automated and relies on manually constructing pre-update runtime state, it does mimic the typical application development scenarios, which are the main interest of this work.

The first part of the experiment determines that updating an application with JRebel without considering the runtime phenomena can indeed crash the application. The application under test is the Space Invaders game, the versions of which differ in how they assign the color of the *Shot object*. In the old version, the color is a constant *Color.YELLOW* returned from a *shot.getColor()* method.

In the new version of the code, *Shot* class has an instance field: *Color color* that is initialized to *Color.GREEN* at the end of the only constructor for the class and returned from the getter. The default value for the color field is *null* and all objects initialized within the new version of the game running have the field Initialized during the regular constructor execution. If the color field happens to be *null*, at the moment the redrawing routine asks for it, the *NullPointerException* is thrown and the program crashes. Both versions of the game work if they start from scratch, and the shot objects flying through the screen have correct colors: yellow and green respectively.

When there are no shots visible on the screen when JRebel updates the application, the update succeeds. The shots that are fired afterward are green. However, if the shots are visible on the game field during the update, the game crashes with the *NullPointerException*, because the shots do not have the color initialized.

If this update is triggered with Genrih performing runtime state analysis, it correctly logs the possibility of the **absent state** phenomenon on the pre-update shots instances. This feedback together with the exception stack trace is sufficient to identify the update of the crash reasonably.

The next phase of the experiment involved significantly updating Space Invaders code, going to another major revision of the game. The functional changes in the update add barrier entities that have to be drawn on the field and make the aliens move and shoot back.

Updating from the initial version of the game to the new shooting version of the game with JRebel brings no visible runtime phenomena. The game proceeds as expected, having the new behavior in place. However, updating the game back to the old version of the code, which has no information about aliens being able to shoot, crashes the program with a *NoSuchMethodError*, because the method *Aliens.fire()*, called from the main game loop is not present anymore.

With the runtime analysis system, the update forward to the shooting version of the code goes in the following fashion. The forward update determines the new instance fields in the *Game* class with the following declarations.

```
private Shots alienShots = new Shots();
private Barriers barriers = new Barriers();
```

The runtime analysis shows that there is one *Game* object on the heap, so the update is postponed due to possible absent state on the Game object after the update. However, since after the update, the application does not crash this false positive feedback is easily ignored.

The downgrade from the shooting version of the game to the basic one crashes with the *NoSuchMethodError* and the notification from Genrih: "Threads are currently running method *Game.gameLoop()* that is changed. An unpredictable update can happen". Together with the *NoSuchMethodError* stack trace, that originates in the *Game.gameLoop()* method, this information is sufficient to consider the DSU being responsible for the crash, not the application logic.

An important additional observation is that, however, most of the time some thread is executing the *Game.gameLoop()* method, there is a small window of time, during the game tick, when it is not on the stack. Then the update proceeds without triggering errors. The dependency of the updates on the current runtime state and given that Genrih can predict if the update will cause no runtime phenomena opens possibilities to stall the update process until the runtime state changes, so the runtime phenomena are impossible. We discuss such advancements of Genrih in the discussion section.

The following updates to even more complicated Space Invaders versions occurred similarly. Without the runtime analysis, adding features and state to the code are handled by existing JRebel well. The downgrades often result in the *NoSuchMethodErrors* caused by calling methods that no longer exist in the new version from the old version of the method still running during the update. Which is the result of not checking the if the updated code is currently actively executed on in the program.

The experiment shows that the designed system provides immediate feedback on the runtime phenomena using an existing stock DSU solution. This feedback and the nature of errors originated in the runtime phenomena clearly indicate that these errors are due to updating the application rather than the code itself.

Debugging the issue with such feedback that strongly suggests the origin of these errors in the updating process is more straightforward than without it.

Rubah cannot update an arbitrary Java application out of the box; the code has to be retrofitted to be aware of Rubah, including the use of Rubah threads and manually inserted markers for places where the code can be updated. Allegedly, the changes to the programs are minimal, the original research paper for Rubah has the programmer's effort estimates for some Java applications: H2 database, Voldemort key-value store, and the Jake2 video game. Retrofitting first two require patches of about 680 lines for each version to make it Rubah aware. Jake2 update is simpler, measuring at around 80 lines of patch per version. It makes it harder to evaluate Rubah on the same set of the updates as JRebel. For the sake of experiment, and to show the generality of the approach, we ran a set of synthetic tests for Rubah on trivial Java applications. The tests included the changes: adding a new instance field without the default value to the objects held in memory, adding new instance fields with the default values, adding static fields with the default values specified, adding new methods, changing the signatures of the running methods, changing class hierarchy and deleting existing classes.

The experiment setup was synthetic, the main class that starts a Rubah thread which loops over a counter and calls a method in the class under update.

The result of the updates is that any data-manipulating change is not handled by Rubah out of the box with the state migrations and results in the absent state and program crash with the NullPointerExceptions. Genrih's integration with Rubah works and produces the notifications about the pre-update objects existing on the heap and that accessing the new fields on them could result in a NullPointerException.

Another side effect of Rubah's approach is that the active method checks never produce any notifications because Rubah stops all running Rubah threads prior the update with a consequence that there are not threads running any code which is updated.

All in all Genrih's integration with Rubah shows that it is capable of predicting possible runtime phenomena, requires almost no changes compared to the integration with JRebel, and does not interfere with Rubah's ability to execute the updates.

In addition to these synthetic tests, we ran the benchmarks on H2, Voldemort and Jake2 applications that were used for assessing Rubah's performance in the original paper. Since these applications were manually chosen and retrofitted for Rubah, the updates do not produce runtime phenomena. However, they show that Genrih does not interfere with updating large long-running Java applications.

More detailed analysis of the experimental results is presented later in the discussion section.

5 Performance Evaluation

This chapter describes the performance overhead of automatically determining the applicability of the state transfer functions of a DSU during the dynamic update. The designed system:

– requesting the list of classes involved in the update,
– running the diff on the old and the new version of the classes to find the exact changes,
– querying the runtime state analysis engine about the current runtime state for possible runtime phenomena caused by the changes (for every changed class).

Requesting the changed classes and diffing the result can be done in parallel with running the application, so the impact of these actions is negligible and the complexity of the operations is linear in the number of classes changed.

Performing the state analysis is more complex. First of all, it must happen when the application is paused so that the state will remain unchanged between the analysis and the actual moment of the update.

The runtime state analysis engine offers the following API for the queries:

```
boolean isClassLoaded(String className)
boolean hasInstances(String className)
boolean hasFieldInitializer(Class klass, String fieldName)
boolean hasNonDefaultFieldValues(String className, String fieldName)
boolean isMethodRunning(String classname, String methodName)
```

This chapter focuses on analyzing the performance of the implementation of the runtime state analysis engine used in the experiment. The main measurements indicate how much time do individual calls to these methods take and the results can be extrapolated to estimate how much time can a single analysis run take. Given that the number of possible queries is limited and is linear in the number of changed classes, the extrapolation is straightforward.

The machine where the benchmarks were run has the following configuration: MacBook Pro (Retina, 13-in., Early 2013) with the 2.6 GHz Intel Core i5 processor, 8 GB 1600 MHz DDR3 memory, and a flash storage hard drive. Java version “1.8.0”, Java(TM) SE Runtime Environment (build 1.8.0-b132), Java HotSpot(TM) 64-Bit Server VM (build 25.0-b70, mixed mode) was used to perform the experiments.

The benchmark was run from a JVM process with the heap size of 1GB. During the benchmark, about 70% of the heap was filled with the objects of dynamically generated classes to model the real world performance. We also started 32 background threads to provide the load for the isMethodRunning query comparable to a real-world use.

To perform benchmarks we utilized Java Microbenchmark Harness (JMH), [20]. The benchmark was configured to measure the average time of the execution of an operation based on 10 sample properly warmed runs. Table 2, originally

Table 2. Runtime queries benchmark.

Benchmark	Score	Error	Units
isClassLoaded	0.001	±0.000	ms/op
fieldInitializer	0.761	±0.062	ms/op
hasInstances	1.430	±0.343	ms/op
nonDefaultFields	1.513	±0.306	ms/op
isMethodRunning	0.099	±0.014	ms/op

produced for the shorter version of the current paper [15], shows the output of a random run of the benchmark using ten iterations.

The time growth is linear of the number of the instances the system has to traverse. For a million of active object instances, it takes consistently under 60 ms. Varying heap size did not influence the timing on the heap sizes up to 3 GB.

The results suggest that the overhead of running a performance check on an incrementally small update to the application code is sub-second. During the experiment with the Space Invaders game, the updates were postponed by 500 ms if the runtime state does not allow to perform it immediately. The analysis did not noticeably slow down the user experience.

Since we have the integration with Rubah whose original paper features performance benchmarks for long-running Java applications H2 database, Voldemort key-value store, and a video game Jake2, we executed the fast versions of H2 and Voldemort benchmarks on a MacBook Pro, 13-in., 2016, with a 3.1 GHz Intel Core i5 processor, and 16 GB of 2133 MHz LPDDR3 memory.

The main goal of the benchmarks is to assess the performance overhead of the application after Rubah applied an update, but they also time the update pause to assess the performance of the novel Rubah parallel and lazy state transformations. Based on the results of the benchmark runs Genrih does not increase the average pause time of the update dramatically, the overhead of running Genrih analysis is approximately 10% of the pause time on small workloads and less than 1% of the larger heaps. The main source of this overhead is that Genrih by default is configured to wait 500 ms and perform the second runtime state analysis to see if the application state changes. In Rubah's case, it almost always does not change, since the application threads are already stopped by Rubah. However, even 10% update time overhead is appropriate for the development time DSU and is a great tradeoff for the additional information about the safety of the update.

6 Discussion

The current research concentrates on the design of a runtime state analysis system for the JVM that can automatically determine if the state transformation functions of a given DSU solution can satisfy a given update. It does so by analyzing the individual changes that the update consists of and querying the runtime state to verify if the declared capabilities of the DSU can handle the scenario at hand.

The proposed system primarily tackles updates performed during development, where the changes to the application are frequent and typically smaller than the difference between two releases of the application. The main benefit of such system is that the developer after introducing the change would likely run the exact piece of code that was changed to verify the correctness of the introduced functionality. As such, any incompatibilities in the runtime state representations in the old and new code have a much higher chance of being stumbled upon and producing a runtime phenomena of the update.

Sometimes the inability to apply an update dynamically can be noticeable. The incomplete update can lead to application errors or crashes. Others, however, are subtle and have less obvious consequences. The errors require developers to investigate the code base to determine the cause of the occurring behavior. That is counterproductive to the core idea of the DSU for the development environments, which is to save time.

In the previous section, we showed a series of dynamic updates of a sample Java application, the Space Invaders game, that illustrated two things. First, the runtime phenomena occur during the dynamic updates, and the updating systems can lack the sophistication to distinguish a safely applicable update from those that will result in a system crash or invalid runtime state of the application. Since the problem of automatic state transformations is not solvable in the general case, [1], any system not requiring manual intervention of the programmer has blind spots for the particular changes and will break the update. This also one can always design an application whose behavior after being updated differs from the behavior of a clean run of the new version. One straightforward way to do this is to rely on the application level data to differentiate between the behaviors. For example, to kill the application process if a certain marker class is already loaded. The old version of the code will then load the class, and proceed to wait for the user input. The new version of the application checks if the marker class is already loaded and decides whether to kill itself. The only way to solve issues like that is to specify the state transformation functions manually. However, it is imperative to avoid the manual intervention from the developer during the updates, since it removes all the time-saving benefits of the dynamic updates.

Second, that our proposed state analysis engine can predict the applicability of the update by querying the runtime state efficiently.

Without the runtime analysis system, even a minor update can result in application crashes as the experiment section showed in the example of dynamic updates applied both with JRebel and Rubah.

The non-deterministic nature of such errors, due to runtime phenomena being dependent on the runtime state at the moment of the update complicates debugging and makes verification of code correctness time-consuming work.

The runtime analysis and feedback system described in this paper, can significantly decrease the frustration of encountering unexpected behavior in the updated application.

Indeed, as the experiment showed, we can determine that the state transformation functions do not satisfy the update at hand. Additionally due to the knowledge of what changes the update consists of and which objects are on the heap or active methods on the stack might produce the runtime phenomena, we preventively notify the developer about the upcoming errors, reducing how unexpected these are.

The list of runtime phenomena the system can predict covers the changes to Java programs previously identified to be capable of producing runtime phenomena: adding or removing static and instance fields of the objects, changing actively running methods or the hierarchy of the loaded classes. The updates performed during the experiment show several significant runtime phenomena types and show that we can apply the knowledge of the capabilities of the state transformation functions of a particular DSU solution with regards to these.

To utilize the described system with an arbitrary DSU, one needs information regarding which individual class changes are supported by the default state transformations of the DSU. In this work, we implemented integration with two DSU systems JRebel and Rubah. To support both systems we needed to introduce very minor changes to the proposed system's code, mostly to utilize the default DSU logging framework for the notifications and error messages.

The experiments showed that both JRebel and Rubah updates can cause runtime phenomena and that Genrih's runtime state analysis system can predict several classes of the runtime phenomena as possible before the update is applied. The conservative approach to predicting runtime phenomena can lead to false positive notifications for both systems, but it is conservative enough to work with both JRebel and Rubah without requiring to tweak it for the particular DSU at hand.

Additionally, the proposed system can potentially be configured to postpone the updates from happening until the runtime phenomena are not possible: until the runtime state transformations are manageable by the DSU at hand.

This approach will use the runtime analysis to determine if the update might lead to the phenomena and apply only safe updates. Otherwise, the update is not continued and rescheduled after a short delay. When a window of opportunity appears when the runtime state changes and the state transformation capabilities of the DSU fit it better.

The exact details of such setup require more research. Using the proposed system to analyze the practical data for applying development time DSU solutions will provide the necessary experimental data for the research of the production systems.

The ability to predict the runtime phenomena by observing the current runtime state of the application allows us to make the daunting task of developing DSU friendly applications easier. Moreover, the ability to identify updates that are complicated for the DSU solutions can be used to collect a corpus of update scenarios for a comprehensive DSU benchmark for Java programs. The existing research of the unified DSU benchmarks concentrates on the updates of the production systems written in lower level languages.

Dynamic Software Update is a complex problem, so any advancement in making it more widespread is a good step forward. Which makes the current work of enhancing the availability of DSU solutions that provide essential information about the predictability and applicability of an update significant.

7 Related Work

To provide the context for this paper, in this section we list some prior work on dynamic software updates focusing on the alternative DSU approaches or on the work which focuses on the safety of the updates.

The field of analyzing the safety of the dynamic updates is not particularly new, but the work is mostly focused on safeguarding the updates to the production and mission-critical systems. Such goal requires rigorous proof of the safety criteria and is not focusing on making the updating system easy and fast to use.

In the paper which gave the direct inspiration to the current work, Gregersen and Jørgensen identified the changes to the Java programs which can lead to runtime phenomena after the updates [14].

Bazzi et al. researched the state mapping problem for DSU and tried to limit the problem of the general DSU to make automatic DSU solutions practically possible, [21].

Zhao et al. devised an automated static analysis system, [13], that suggests safe points in a program, which can be run on the arbitrary applications without prior specification of the original safe points.

There also exist multiple dynamic software update solutions for the Java programs, [3,12].

The DSU systems we integrated Genrih with are Rubah [5] and JRebel [4]. Rubah uses bytecode rewriting to enable dynamic update on the stock JVM. The main novelty of Rubah was the lazy state transformation approach to speedup the update times. On the downside, Rubah requires programmers effort to make applications updatable, which makes it less attractive for the development time updates.

JRebel is a state-of-the-art development time DSU solution for Java capable of reloading all changes to the application seamlessly, [4]. However, JRebel does not implement any non-trivial measures to ensure runtime phenomena free updates, which made JRebel a perfect candidate for a stock DSU solution to be enhanced with Genrih.

Rather than trying to devise a generic solution capable of updating the running application at any moment of time, another approach to making dynamic updates safe is to allow developers to specify the "quiescence" safe points in the application code where it is safe to apply the updates. For some application architectures, like the event-driven systems, such approach, is incredibly straightforward and does not require extensive changes to the application architecture.

Hayden et al. integrated multi-threaded quiescence into Kitsune and experimented with updating a real-life event-driven system to evaluate the performance of the approach. The results suggest that in an event-driven system it is relatively easy and fast to catch all the threads into a safe point, [2].

Another Hayden et al. work focuses on techniques for establishing the correctness of the dynamic updates, [6]. They present a methodology for automatically verifying the correctness of dynamic updates using specifications provided by developers. The main approach lies in the provably correct merge transformation of the old and new versions of the code into a merged entity; that is later analyzed both statically and using a symbolic executor for the correctness of the dynamic update.

Giuffrida et al. introduced a system for live updates that uses time-traveling snapshot techniques to maintain the balance and transfer the runtime state back and forth between two versions of the code, [22]. Their approach also trades the time for an update for its safety, and not particularly applicable to the development time DSU.

The main focus of the existing research on DSU seems to be concentrated either on making the production system updates more timely and safe by introducing manually or statically determined safe points or encouraging to create the DSU aware applications by following a certain programming approach. However, to the best of our knowledge, inspecting runtime state of a statically typed object-oriented runtime to determine if an update is safe has not been previously discussed in details.

8 Conclusion

Previous research has determined that applying updates to the running code on the fly can result in the visible application-level side effects due to the runtime phenomena occurring after the update. The main reason for the phenomena to occur is breaking application assumptions about the runtime state because the update cannot automatically convert all the runtime state.

In this paper, we have proposed Genrih, a system that enhances an existing DSU system for statically typed programming languages with capabilities to analyze whether the runtime state of an application satisfies queries for avoiding unwanted runtime phenomena.

The main contribution of the paper is the design and the implementation of a runtime state analysis system that can automatically decide if the state transformation functions of the underlying DSU is sufficient for the current update. It is orthogonal to the actual solution performing the dynamic update and depends on just a handful of information about the update mechanics. This is underpinned by the fact that Genrih was tested with two major DSU systems, JRebel and Rubah.

Genrih operates by combining the knowledge of the exact changes an update consists of, as well as the current runtime state to determine what runtime phenomena can be observed if the update is applied immediately. We provided a construction of a mapping for the types of changes to a Java program that has been identified as capable of surfacing the observable runtime phenomena. We implemented a prototype of the system that is capable of predicting the possibility of these phenomena. The system prototype was integrated with both JRebel and Rubah dynamic updating functionality for Java applications to show the generality of the approach.

The main result is that it qualifies the DSU friendliness for the particular updates. At the same time, it provides preemptive feedback describing the runtime phenomena possibly caused by the update and their origins. Hence, Genrih serves both as an educational tool in the sense that developers will become aware of certain pitfalls but also as a mean to eliminate a major time waste the existing dynamic software update solutions which operate primarily at development time are susceptible to – debugging errors introduced by the runtime phenomena of the dynamic updates.

References

1. Gupta, D., Jalote, P., Barua, G.: A formal framework for on-line software version change. IEEE Trans. Softw. Eng. **22**, 120–131 (1996)
2. Hayden, C.M., Saur, K., Hicks, M., Foster, J.S.: A study of dynamic software update quiescence for multithreaded programs. In: Proceedings of the 4th International Workshop on Hot Topics in Software Upgrades (HotSWUp), pp. 6–10. IEEE Press, Piscataway (2012)
3. Subramanian, S., Hicks, M., McKinley, K.S.: Dynamic software updates: a VM-centric approach. SIGPLAN Not. **44**, 1–12 (2009)
4. Kabanov, J., Vene, V.: A thousand years of productivity: the JRebel story. Softw.: Pract. Exper. **44**, 105–127 (2014)
5. Pina, L., Veiga, L., Hicks, M.: Rubah: DSU for Java on a stock JVM. In: Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications, (OOPSLA 2014), pp. 103–119. ACM, New York (2014). https://doi.org/10.1145/2660193.2660220
6. Hayden, C.M., Magill, S., Hicks, M., Foster, N., Foster, J.S.: Specifying and verifying the correctness of dynamic software updates. In: Joshi, R., Müller, P., Podelski, A. (eds.) VSTTE 2012. LNCS, vol. 7152, pp. 278–293. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27705-4_22
7. Hayden, C.M., Smith, E.K., Hardisty, E.A., Hicks, M., Foster, J.S.: Evaluating dynamic software update safety using systematic testing. IEEE Trans. Softw. Eng. **38**, 1340–1354 (2012)
8. Hayden, C.M.: Clear, correct, and efficient dynamic software updates. Ph 3543 (2012)
9. Arnold, J., Kaashoek, M.F.: Ksplice: automatic rebootless kernel updates. In: Proceedings of the 4th ACM European Conference on Computer Systems, EuroSys 2009, (Nuremberg, Germany, April 2009), pp. 187–198. ACM, New York (2009)
10. Erlang: Erlang reloading documentation (2017). http://www.erlang.org/doc/reference_manual/code_loading.html. Accessed 17 May 2017
11. Würthinger, T., Wimmer, C., Stadler, L.: Dynamic code evolution for Java. In: Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java, PPPJ, pp. 10–19. ACM, New York (2010)

12. Gregersen, A.R., Jørgensen, B.N.: Dynamic update of Java applications - balancing change flexibility vs. programming transparency. J. Softw. Maint. Evol. **21**, 81–112 (2009)
13. Zhao, Z., Ma, X., Xu, C., Yang, W.: Automated recommendation of dynamic software update points: an exploratory study. In: Proceedings of the 6th Asia-Pacific Symposium on Internetware (INTERNETWARE 2014), pp. 136–144. ACM, New York (2014). https://doi.org/10.1145/2677832.2677853
14. Gregersen, A.R., Jørgensen, B.N.: Run-time phenomena in dynamic software updating: causes and effects. In: Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution (IWPSE-EVOL 2011), pp. 6–15. ACM, New York (2011). https://doi.org/10.1145/2024445.2024448
15. Šelajev, O., Gregersen, A.: Using runtime state analysis to decide applicability of dynamic software updates. In: Proceedings of the 12th International Conference on Software Technologies, pp. 38–49 (2017)
16. Gregersen, A.R.: Implications of modular systems on dynamic updating. In: Proceedings of the 14th International ACM Sigsoft Symposium on Component Based Software Engineering, (CBSE 2011), pp. 169–178. ACM, New York (2011). https://doi.org/10.1145/2000229.2000254
17. Šelajev, O., Raudjärv, R., Kabanov, J.: Static analysis for dynamic updates. In: Proceedings of the 9th Central and Eastern European Software Engineering Conference in Russia, (CEE-SECR 2013). ACM, New York (2013)
18. Oracle: JVMTI documentation (2017). http://docs.oracle.com/javase/8/docs/platform/jvmti/jvmti.html#whatIs. Accessed 18 May 2017
19. Pina, L.: Rubah source code (2014). https://github.com/plum-umd/rubah. Accessed 22 Nov 2017
20. Shipilev, A.: Java microbenchmark harness (2017). http://openjdk.java.net/projects/code-tools/jmh/. Accessed 17 May 2017
21. Bazzi, R.A., Makris, K., Nayeri, P., Shen, J.: Dynamic software updates: the state mapping problem. In: Proceedings of the 2nd International Workshop on Hot Topics in Software Upgrades, (HotSWUp 2009), Article no. 7, 2 p. ACM, New York (2009)
22. Giuffrida, C., Iorgulescu, C., Kuijsten, A., Tanenbaum, A.S.: Back to the future: fault-tolerant live update with time-traveling state transfer. In: Proceedings of the 27th USENIX Conference on Large Installation System Administration (LISA 2013), pp. 89–104. USENIX Association, Berkeley (2013)

Software Systems and Applications

Identifying Class Integration Test Order Using an Improved Genetic Algorithm-Based Approach

Istvan Gergely Czibula, Gabriela Czibula, and Zsuzsanna Marian(✉)

Department of Computer Science, Babeş-Bolyai University,
M. Kogalniceanu Street, Cluj-Napoca, Romania
{istvanc,gabis,marianzsu}@cs.ubbcluj.ro

Abstract. *Software testing* is a very difficult activity, representing a large part from a software system's development process. Within the *class-based integration testing* methodology, determining the order in which the application classes have to be tested is of major importance for reducing the testing time and cost. The Class Integration Test Order (CITO) problem deals with identifying the testing order of classes which minimizes stub creation effort, and subsequently testing cost. In this paper we propose an efficient approach using a genetic algorithm with stochastic acceptance for determining the *class integration test order* which minimizes the *stubbing effort* needed during the class-based integration testing. In our proposal, we estimate the stub creation complexity by allocating weights to different types of dependencies between the classes in the software system's Object Relation Diagram. Four synthetic examples and six software systems often used in the CITO literature are used as case studies for experimentally evaluating our proposal. The effectiveness of our approach is confirmed by the obtained results that outperform the existing related work which provide experimental results on the case studies considered in this paper.

Keywords: Integration testing · Class integration test order
Genetic algorithm

1 Introduction

Search-Based Software Testing is an important subfield of the *Search-Based Software Engineering* domain in which different metaheuristic search algorithms are used for solving software engineering related problems. Besides search algorithms, other approaches and machine learning algorithms have been proposed for solving NP-complete software engineering problems, including testing-related activities. Machine learning approaches are appropriate for such problems, since they offer adaptive automated and semi-automated solutions in situations characterized by large and complex problem spaces.

E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 163–187, 2018.
https://doi.org/10.1007/978-3-319-93641-3_8

Class-based integration testing is a systematic testing technique applied when the application classes of a software are integrated in the final software system. The classes are integrated sequentially and after adding each class the obtained system is tested. If no errors have been found during testing at some point in the integration, then the next application class will be integrated. An important problem during integration testing of object-oriented software systems is the one of deciding the order in which the application classes should be integrated in the final software, called the *class integration test order* (CITO) problem [1].

In most situations, there is a dependency relation between the application classes, namely a class may require another class to be available before it can be tested. In cases when dependency cycles exist among the application classes from a software system, at least one dependency has to be broken and a *stub* for emulating the behavior of the required class has to be created [2]. If, at a particular step during class-based integration testing, one adds a class which depends on another application class that has not been integrated yet, a simulation of that class is necessary. This is done by creating a *stub* for the required class, more precisely a dummy class that replaces the required one and simulates its behavior. *Stubs* are those parts of a software system that are built for simulating components of the software which are not developed or unit tested yet, but are needed to test classes that depend on them [1]. There is a difference between *specific* and *generic* stubs. A *specific* stub replicates only the class functionalities for a specific client class, while *generic* (realistic) stubs reproduce all functionalities that the original class can provide. Therefore when a class is used by many client classes we will need only one generic stub, but as many specific stubs as the number of client classes. Since stub creation increases the cost of the integration testing process, it is essential to reduce stubbing cost by determining a class order for integration testing that minimizes the overall stubbing effort.

The CITO problem does not cover aspects related to the actual creation of stubs nor does it approach the problem of *test case effectiveness*. The main objective of the CITO problem is to reduce the number of stubs needed, not to increase early bug detection. Software developers are still responsible for creating stub classes that closely model the effective class to be stubbed.

There are numerous strategies proposed in the literature for solving the CITO problem with the aim of minimizing the *stubbing effort* required during the integration process. The *stubbing effort* estimates the cost of creating the stubs needed during the integration testing. It can be computed either as the number of needed stubs or considering measures related to *coupling*, number of attributes and methods or the complexity of the methods that need to be replicated. Most of the solutions existing in the literature for the CITO problem can be divided in two categories: graph-based approaches and genetic algorithm-based approaches [3]. The graph-based approaches consider the Object Relation Diagram (ORD) which represents the classes and the relationship between them in object-oriented software systems. As shown by Briand et al. in [4], the graph-based solutions for solving the CITO problem are very hard to be adapted to take into account several factors related to the stub creation (like constraints connected to organizational or contractual reasons, number of calls or distinct methods). Another shortcoming of graph-based solutions is that they consist of identifying and removing Strongly Connected Components that maximize the number of broken cycles. However, there are situations when breaking only one dependency is more costly than breaking two dependencies and thus, the obtained solutions can be sub-optimal [4]. The authors also show that the *genetic algorithms*-based approaches are able to mitigate the above mentioned drawbacks.

A more general approach of the *class integration test order* determination problem is the *test case ordering* (TCO) problem. The TCO problem can be formulated as the problem of constructing a sorted collection of test cases, collection that reflects an optimal execution order of the test cases, in order to optimize a certain objective function (criterion).

In this paper we are approaching the CITO problem as a combinatorial optimization problem, with the goal of determining the order in which the application classes should be tested for minimizing the total cost of *stubbing.* We consider the *stubbing* cost as the effort for creating the specific stubs needed during the class-based integration testing. The complexity of creating a stub is estimated by assigning weights to different types of dependencies (i.e., aggregation, association, inheritance) in the software system's Object Relation Diagram.

We have proposed in [5] an efficient approach for optimizing the *class integration test order* using a genetic algorithm with stochastic acceptance. Our proposal was based on a static analysis of object-oriented software systems and improved the existing strategies based on genetic algorithms for finding a solution for the CITO problem. A general theoretical model that can be applied both for generic stubs and specific stubs with different weighting strategies was also provided in [5].

In this paper, we extend the experimental evaluation and the analysis from our previous approach from [5]. Experiments are performed on four synthetic examples and on other six case studies often used in the literature for the *class integration test ordering* problem. The potential of our proposal is confirmed by the obtained results which are better than those of existing related work which provide experimental results on the case studies considered in this paper. We mention that the approach introduced in this paper is general. Even if it is presented to solve the CITO problem, it can be easily extended for solving the more general TCO problem.

The paper is organized as follows. We start by reviewing in Sect. 2 existing approaches which provide solutions for the CITO problem considering weighted stubs and give experimental results on the case studies that are considered in this paper. Our approach based on a genetic algorithm with stochastic acceptance for solving the CITO problem is introduced in the Sect. 3. Section 4 describes the case studies used for evaluation and also provides the experimental settings and results. An analysis of our approach as well as a comparison of the results obtained by our proposal with some state-of-the-art techniques are provided in Sect. 5. Our conclusions as well as several future research directions are presented in Sect. 6.

2 Related Work

In this section we will present a short overview of existing approaches for the CITO problem, focusing mainly on the ones that, like our approach, consider that not every stub has the same complexity. Most of these approaches build a graph, called Object Relation Diagram (ORD), where nodes represent the application classes and directed edges represent the relationships between these classes. Edges often have labels that represent the type of the relationship between the two classes. The number of relationships can be different from one approach to another, but the most frequently used relations are inheritance, aggregation and association.

If the ORD contains no cycles then a simple topological sorting can give the integration order. For such systems, a bottom-up integration strategy can be used, and no stubs are needed. But in most software systems there are cyclic dependencies between the classes in the ORD as shown by Melton and Tempero in [6] where a study was conducted on 78 Java software systems with different sizes (from 17 to 11644 classes). The authors concluded that almost all systems contained cycles, moreover, about 85% of them contained strongly connected components of at least 10 classes. In order to eliminate the cycles, either edges or nodes have to be removed from the ORD. When an edge is removed a *specific* stub will be needed to simulate it, however when a node is removed - together with all incoming and outgoing edges - either one single *generic* stub has to be created or a *specific* stub for every incoming edge connected to the removed node.

The first paper that considered the CITO problem was written by Kung et al. [7]. They consider an ORD with inheritance, aggregation and association relations. The first step of their approach is to transform it into an acyclic one, by first replacing clusters of mutually reachable nodes with one single node. A topological sorting of this acyclic graph will give the *major level* of nodes. For finding the *minor level* of nodes, the order inside the clusters, association relations are removed, since every cycle has to contain at least one association edge.

Major and minor level numbers are used by Tai and Daniels as well [8], but they are computed differently. For assigning major level numbers, only the inheritance and aggregation relations of the ORD are considered and a Depth First Search (DFS) is performed. Minor levels are assigned to nodes that have the same major level. Strongly Connected Components (SCC) are identified for the nodes belonging to the same major level, and to each association edge e a weight is assigned as the sum of the number of incoming edges to the source node of e and the number of outgoing edges from the target node of e. The edge with the highest weight is removed, because it has a higher chance of breaking many cycles. This process is repeated until no cycles are left.

Le Traon et al. propose in [9] a solution for the CITO problem based on a *test dependency graph* (TDG) which is constructed by mapping the UML diagram to a graphical model of dependencies. A TDG is more detailed than an ORD, because it can contain method-to-class and method-to-method relations as well.

The authors propose an adaptation of the Bourdoncle algorithm for determining the testing strategy. The proposed solution has a time complexity of $O(n^4)$ and offers an heuristic which provides results very close to the optimal one.

TDG is used by Le Hanh et al. as well in [10] where they present two integration strategies, a graph-based one and a genetic algorithm (GA) based one. For the graph-based approach, called *Triskell*, they find the node that participates in the maximum number of cycles and remove it (consider that it will be stubbed). This produces one generic stub, and a specific stub for each incoming edge into the node. Relation types are considered only if two nodes participate in the same number of cycles, in this case the node with more association relations participating in cycles is removed. They repeat the process until no cycles are left. The GA approach considers only the number of stubs, no relations are considered.

In [1] Briand et al. propose a graph-based approach, which identifies SCC in the graph, and for each association edge in each SCC computes a weight which is similar to the weight computed by Tai and Daniels, but instead of taking the sum, for an edge they take the product of the number of incoming edges to the source node and the number of outgoing edges from the target node. They remove the edge with the highest weight and continue until no cycles are left. In [11] Briand et al. proposed a Genetic Algorithm based approach as well. They use constraints to make sure that inheritance and aggregation relations will not be broken (they specify partial ordering of nodes based on these relations) and compute weights for association edges. The weight of an association relation depends on the complexity of the class represented by the target node, and this complexity depends on the number of methods and/or number of attributes of the class.

Mao and Lu present in [12] an approach, called AICTO, which breaks only association edges, but tries to estimate the complexity of the association relation by computing weights for these edges. The authors take into consideration how many methods are called and attributes are accessed, but also the number of cycles in which the edge participates (with a formula similar to the one from [1]).

While previous approaches considered weights mainly just for association edges (since inheritance and aggregation edges are never removed), the approach presented by Malloy et al. in [13] considers weights for 6 different relations: association, composition, dependency, inheritance, ownedElement, polymorphic. They use the ORD and find SCC in it. For each SCC they compute the weight of the edges and remove the edge with the minimum weight. For the experimental evaluation they use 7 case studies of different sizes and two different sets of weights. The only difference between the two sets of weights was the weight assigned to inheritance edges. In the first set of weights, inheritance has weight 2, which is a low value, making it probable that inheritance edges will be removed. In the second set, inheritance has a weight of 100 which makes removal of inheritance edges very unlikely. They conclude that in the situations when no inheritance edges are removed, approximately twice as many stubs are needed. Similar experiments were performed by Kraft et al. in [14] and the same conclusions were reached regarding the removal of inheritance edges. For one case study made of 236 classes, removing 9 inheritance edges reduced the total number of required stubs by 600, compared to a version where inheritance edges were not allowed to be removed.

Another graph-based approach is the one presented by Abdurazik and Offutt [15]. The novelty in their approach is that they consider weights for both edges and nodes in the ORD (though the weights for the nodes are computed considering the weights for the edges). They consider 9 different relations between classes and compute the weight of an edge based on several measures of coupling. Their algorithm computes for each edge a Cycle to Weight Ratio, which considers both the number of cycles that include that edge and the weight of the edge. The edge with the maximum CWR is removed, and the process is repeated until no cycles are left.

Bansal et al. present an approach which is based on the approach presented by Malloy, but they introduce two new relation types specific for C++ applications: friend coupling and exception coupling and define weights to them: 25 and 5 [3]. They also present an overview of existing graph-based and genetic algorithm-based approaches.

Guizzo et al. propose in [16] a Hyper-heuristic HITO for the Integration and Test Order (ITO) problem, for determining a sequence to integrate and test software units with a minimal cost. HITO combines a crossover operator and a mutation operator used by Evolutionary Algorithms (EAs) for selecting the best heuristic [16]. The work of Guizzo et al. was the first approaching the ITO problem using hyper-heuristics. The authors have shown that the application of multi-objective evolutionary algorithms (MOEAs) may be suitable for the CITO problem, but applying them may be difficult [16].

Mariani et al. present in [17] GEMOITO, a hyper-heuristic based on Grammatical Evolution (GE), which aims to automatically generate a Multi-Objective Evolutionary Algorithm (MOEA) for solving the ITO problem. The goal of the proposed approach is to reduce the effort for choosing, implementing and configuring the search techniques which can be used for solving the ITO problem. The authors emphasize that the proposed hyper-heuristic can generate MOEAs that are statistically better or equivalent to the conventional ones.

An approach that is neither graph-based nor search-based is presented by Zhang et al. in [18]. Similar to Briand, they consider that the complexity of a stub depends on the number of accessed attributes and called methods in the class that is stubbed. For every class they compute a *test cost* and a *test revenue*. The test cost is the complexity of the stubs needed to integrate the given class, while the test revenue is the complexity of the stubs that will not be created later if the given class is integrated now (the complexity of the stubs of classes not integrated yet that depend on the given class). At every iteration the class with the greatest difference between test revenue and test cost is integrated, together with all the classes with zero test cost (classes not requiring stubs). After every iteration test cost and revenue are recomputed. Zhang et al. show that the results produced by this approach are comparable and sometimes better than results of existing approaches and the algorithm runs faster than graph- or search-based approaches.

3 Methodology

In this section we introduce our proposal for optimizing *class integration test order* using a genetic algorithm (GA) with stochastic acceptance. GAs are used due to their flexibility and applicability in successful solving of a large variety of optimization problems.

Our approach is based on a static analysis of object-oriented software systems and on computing the *stubbing effort* as the cost of creating the specific stubs needed during the integration testing. We consider that the cost (effort) needed to create a stub depends on the type of the dependency between the classes. Consequently, we will work with *weighted stubs* where the complexity of creating a stub will be computed by assigning weights to different types of dependencies between the application classes. In our model every edge labelled with a given dependency type will have the same weight, i.e., every association relation will have the same weight and every inheritance will have the same weight, etc.

We mention that if the weights associated to the relationships are all equal (see the **equal weighting** scheme from Sect. 4.1), the GA proposed in this section will provide a class ordering which minimizes the number of non-weighted *specific* stubs.

We start by describing in Sect. 3.1 how the class relationships are used in the literature for stubbing, followed by the dependencies and weighting scheme considered in our approach. Section 3.2 presents the main characteristics of genetic algorithms, while Sect. 3.3 introduces the genetic algorithm model proposed for the CITO problem.

3.1 Stubbing Relationships

Kung et al. [7] show that if there are no dependency cycles among classes in the ORD of a software system, the integration order can be simply obtained by performing a reverse topological ordering of classes based on their dependencies [1]. But if cyclic dependencies among classes can be found, most existing strategies propose breaking some dependencies for obtaining an acyclic graph and then applying a topological sorting on it. Breaking a dependency requires a stub for the target class when integrating and testing the source class [1].

Three types of dependencies between application classes in the ORD are considered by Kung et al. [7]: Association/Usage (As), Aggregation (Ag) and Inheritance (I). In order to obtain an acyclic graph, the authors propose the removal of association relationships. They consider that at least one association relationship exists in each directed cycle of an ORD and this type of relation is the weakest one between related classes.

A literature review reveals that many approaches consider that by removing association relationships simpler stubs are created compared to those obtained by selecting aggregation or inheritance relationships [7,8,19]. However, Le Traon et al. [20] propose a strategy allowing to break aggregation or inheritance relations, which may lead to complex stubs. The results obtained by Malloy et al. [13] revealed that the removal of inheritance relationships is more effective for cycle breaking. Kraft et al. [14] consider six different types of dependencies between application classes, some of them specific for the C++ programming language, and use the following relationship between the costs (weights) associated to them: $cost(dependency) = cost(association) = cost(polymorphic) < cost(composition) = cost(ownedElement) \ll cost(inheritance)$.

In our proposal we are considering a Weighted ORD with the three dependency types considered by Kung et al. [7], where each relationship (As, Ag and I) has an associated weight. The weighting scheme reflects the relative effort needed to implement a stub class for a particular client class that has a dependency on the stubbed class.

We will assign different weights to these relationships, in order to test how the relationships influence the total cost of stubbing. A stub class is a replacement for an existing dependency in the system, and we consider that the main factor that influences the effort needed for creating it is the type of the dependency. Three weighting schemes in accordance with the relations defined by Kraft et al. in [14] (DW2, DW3 and DW4 from Table 1) will be considered, in which the *inheritance* (I) relationship is the strongest relationship between the application classes, followed by the *aggregation* (Ag) and then by the *association* (As) relationship which is viewed as the weakest relation between the classes. In order to test the complexity of stubbing the *inheritance* relationship, we will also use a weighting scheme, DW1 from Table 1, in which the strongest relationship is Ag followed by I and As that have the same weight.

Even if our current implementation considers only I, As and Ag relationships between the application classes, our approach can be simply extended to consider other relationships between the classes, such as the coupling between them.

3.2 Genetic Algorithms

Genetic Algorithms (GAs) represent a specific type of metaheuristic optimization techniques which are inspired from the biological processes of evolution and selection in nature. They are mainly used for solving search and optimization problems. Since GAs are based on heuristics, there is no guarantee that they will converge to the global optimum. However, they are able to provide a solution close to the optimal one and prevent the search from falling into local minima [21].

The main idea behind GAs is that a population of individuals adapts to environmental changes over multiple generations, and the fittest individuals from the population are those who survive longer [22].

GAs start with a population of *noInd* candidate solutions, also called *individuals* or *chromosomes*, usually randomly generated. Each individual from the population is characterized by a numerical value called its *fitness*, which indicates how "good" is that individual for solving the considered problem. Over a number of iterations (*generations*) or until acceptable solutions are found, the population is *evolved* using *genetic operators* as follows. A pair of chromosomes is selected (using a *selection* strategy), then we **cross-over** (with probability p_c) the selected pair and form two offspring and lastly we **mutate** the two offspring (with probability p_m) and add the obtained individuals in the new population. At the end of the iterative process, the individual with the maximum fitness from the current population is reported as a solution.

Figure 1 describes the skeleton of a simple GA.

```
Algorithm $GA$ is:
Input:  $noInd$ - the number of individuals from the population
        $p_m$- the probability for mutation
        $p_c$- the probability for crossover
Output: $best$ - the best individual (solution)
Begin
      //a population of $noInd$ individuals is initialized (usually randomly)
   $P \leftarrow initializePopulation()$;
            //repeat until a termination criterion is met (no. of generations, etc)
   While not TERMINATION do
         //the fitness of the individuals from the population is computed
     $computeFitness(P)$;
     //the survivors from the current population are preserved
     $P' \leftarrow survive(P)$;
     //repeat until $noInd$ individuals are created
     While $|P'| < noInd$ do
              @ Select from $P$ a pair ($c_1$, $c_2$) of chromosomes for crossover
           @ With probability $p_c$ cross-over ($c_1$, $c_2$) and form two offsprings

        @ With probability $p_m$ mutate the offsprings and add them in $P'$
     $EndWhile$;
     //the current population is replaced with the new population $P \leftarrow P'$;
   EndWhile
      //the individual with the maximum fitness is reported as a solution
   $best \leftarrow maxFitness(P)$;
End $GA$
```

Fig. 1. The skeleton of a GA [5].

3.3 The Proposed GA Model

Let us consider that the analyzed software system $\mathcal{S}$ is composed by a set of classes $C_1, C_2, \ldots, C_n$. Starting from the ORD graph built for the software system and based on a static analysis of it, we aim to identify an appropriate order in which the application classes should be integrated (and tested) in the final software. The solution is viewed as a permutation of the classes representing the integration order that needs the minimum *stubbing effort*. Consequently, the optimal solution for the CITO problem is viewed as a permutation τ of $\{1, 2, \ldots, n\}$ which minimizes the total *cost* for creating the stubs needed when the classes are integrated and tested in the order τ: $C_\tau = (C_{\tau_1}, C_{\tau_2}, \ldots, C_{\tau_n})$ $(n > 1)$. The *stubbing effort* required for the integration testing of a sequence of classes $C_\tau = (C_{\tau_1}, C_{\tau_2}, \ldots, C_{\tau_n})$ is denoted by $\mathcal{C}ost_{C_\tau}$ and is defined as in Formula (1) [5]:

$$\mathcal{C}ost(C_\tau) = \sum_{i=1}^{n} stub(C_{\tau_i}, C_{\tau_{i-1}}, \ldots, C_{\tau_1}) \qquad (1)$$

where $stub(C_{\tau_i}, C_{\tau_{i-1}}, \ldots, C_{\tau_1})$ represents the cost for creating the weighted stubs for integrating the class C_{τ_i} to the system formed by the classes $\{C_{\tau_{i-1}}, C_{\tau_{i-2}}, \ldots, C_{\tau_1}\}$. This cost is computed by summing the weights associated with the relationships between class C_{τ_i} and all its neighboring classes from the Weighted ORD which were not already integrated.

An individual from the GA population is an integer-valued vector whose length is equal to the number of application classes from the analyzed software system and represents a possible order for integrating the classes during the integration testing. Thus, a candidate solution to the CITO problem is encoded in an individual (chromosome) $ind = (ind_1, ind_2 \ldots ind_n)$ representing a permutation of $\{1, 2, \ldots, n\}$ ($1 \leq ind_i \leq n \quad \forall i \in \{1, 2 \ldots n\}$ and $ind_i \neq ind_j \quad \forall 1 \leq i, j \leq n, i \neq j$).

We define the value of the *fitness function* for a given individual *ind* as in Formula (2) [5].

$$fitness(ind) = Max - \mathcal{C}ost(C_{ind}) \tag{2}$$

where Max represents a large positive constant. Considering the definition of the fitness given in Formula (2), maximizing the fitness of a chromosome *ind* will be equivalent with minimizing the stubbing cost required when the classes are integrated and tested in the order C_{ind}. Accordingly, the components of the fittest individual reported by the GA will give us the class integration test order.

The improvements we propose to the classical GA model are presented in the following.

It is well-known that a limitation of GAs is that they are very sensitive to the initial population. In order to overcome this limitation and to assure a proper exploration of the search space, the initial population is generated using an heuristic which will be described in the following. Given the fact that every chromosome represents a permutation, the dimension of the search space is $n!$ where n is the number of classes in the system. In order to evenly divide the search space we will generate the k-combinations for the set of classes, where k is chosen based on the size of the initial population. For example if we have the set of classes $\{A, B, C, D, E, F, G, H\}$ we can generate 2-combinations: $\{A, B\}, \{A, C\}, \ldots$ in total 28 different sets with 2 elements. We use the k-sets when generating the initial population by creating chromosomes that start with genes according to the generated k-combinations followed by randomly generated genes.

The GA model we propose for solving the CITO problem also considers a *selection* operator based on the *stochastic acceptance* technique [23]. In this algorithm, the widely used *roulette-wheel* selection operator used in the population for reproduction is replaced with the following one: an individual *ind* is randomly selected and this selection is accepted with probability $fitness(ind)/fitness(M)$, where M is the fittest individual (has the highest fitness value). If the individual is not accepted, a new one will be randomly selected and accepted with the given probability and so on until the first accepted individual. Through the *stochastic acceptance* based selection operator, the fittest individual will always be accepted if selected. Several studies in the literature indicate that the selection based on stochastic acceptance performs considerably better than versions based on linear or binary search [23].

In our proposed GA we used a variant of the *Order Crossover 1* operator, known as *C1*, which is specific for using GAs for permutation problems [24]. Basically, a sequence of consecutive genes is removed from the first parent and is directly copied to the child. The remaining genes are placed in the offspring in the order in which they appear in the second parent. From the time performance viewpoint, *C1* is the fastest crossover operator that generates valid chromosomes (preserves the ordering constrains).

The mutation operator is a variant of the *swap mutation*, but we swap a randomly generated gene with its adjacent gene.

We have also used *elitism* in our GA, which means that the next generation will always contain a small proportion ($P_{elitism}$) of the fittest individuals from the current population.

4 Computational Experiments

We provide in this section an experimental evaluation of our GA approach presented in Sect. 3.3 for solving the CITO problem on ten case studies often used in the literature: four synthetic examples and six real-life case studies.

4.1 Parameters Setting

For the GA model proposed in Sect. 3.3 we used our own implementation, without any third party libraries. The following parameters setting will be used for all experiments: the constant Max used in the fitness computation was set to 10000; the number of individuals from the GA population $noInd$ is $2 \cdot C_n^2$; the mutation and crossover probabilities are $p_m = 0.3$ and $p_c = 0.3$; the proportion $P_{elitism}$ used for the *elitism* parameter was considered 0.1 (10% of the population survives from one generation to another). As a termination condition for our GA we used a predefined number of trials depending on the number of application classes, $50 \cdot n$. Regarding the parameters, we tried different values, but no significant difference was observed in the obtained results.

The experiments are conducted in two directions, considering different weighting schemes for the stubs in computing the *stubbing effort*.

1. **Equal Weighting.** In this scheme, equal weights (e.g. 1) are assigned for all stubs, independent of the type of relationship between a client class and the application class to be stubbed. Such a weighting method allows us to determine the number (and list) of non-weighted *specific* stubs. For this scheme, the solution with the minimum stubbing effort is the class ordering with the minimum number of required specific stubs.
2. **Differential Weighting.** In this scheme we assigned different weights for the different types of relations from the ORD. The values for the weights were selected after analyzing the similar literature which assigns weights for

the stubs [13], as well as our software development experience. In order to analyze how different values for the weights influence the number of required stubs, we have decided to use four sets of weights described in Table 1.

Table 1. Differential weighting schemes.

Differential weighting scheme	Weights		
	Inheritance	Association	Aggregation
DW1	5	5	20
DW2	30	5	20
DW3	100	5	20
DW4	100	20	20

For the first set of weights (DW1) we decided to assign a larger weight to aggregation and equal weights for inheritance and association. The second set of weights DW2 is the one used in [5] and was chosen based on the values found in the literature and the general principle that inheritance relations are the hardest to stub and association relations are the easiest. For the third set, DW3, we increased the cost of inheritance, similarly to the experiments presented in [13]. In the fourth set of weights, DW4, we assigned the same cost to association and aggregation and a much larger cost to inheritance. A similar weight set was used in [14]. The main goal of the last three sets of weights is to see how the cost of inheritance is related to the number of stubs, since some results from the literature show that assigning a high weight to inheritance will lead to a lot more stubs [13,14].

The ratio between the values of the weights from every set shows the difference between the complexity of stubbing different relations. In case of DW3, for example, the weights show that we consider that it is 4 times more complicated to stub an aggregation relation than an association (or that we consider that creating 4 stubs for association is as complex as creating one stub for aggregation) and that stubbing inheritance is 20 times harder than stubbing association and 5 times harder than stubbing aggregation.

4.2 Example

We start our experiments with a simple example illustrating how our approach works. In our example there are four application classes A, B, C and D, as shown by the ORD from Fig. 2. On Fig. 2, a directed edge between two application classes indicates that a relationship exists between them and the edge is labeled with the type of relationship: I for *Inheritance*, As for *Association* and Ag for *Aggregation*.

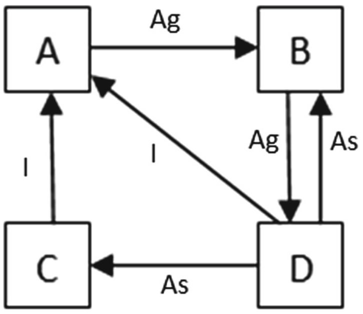


Fig. 2. The sample software system. The directed edges between the classes illustrate a relationship between them.

We consider in the following only the differential weighting scheme DW2 for the relationships between classes. Using a brute force approach, we determined all the possible permutations of the 4 application classes from the system illustrated in Fig. 2 and the total cost $\mathcal{C}ost$ for creating the weighted stubs for integrating the classes in the order corresponding to each permutation. Table 2 presents each permutation with its corresponding $\mathcal{C}ost$ and the required number of stubs. The permutation corresponding to the minimum stubbing cost is highlighted.

Table 2. All the possible permutations, with their associated stubbing cost $\mathcal{C}ost$ (Eq. (1)).

No.	**Class order**	**Stubbing cost**	**Stubs**	**No.**	**Class order**	**Stubbing cost**	**Stubs**
1	$ABCD$	40	2	**13**	$CBAD$	50	2
2	$ABDC$	45	3	**14**	$CBDA$	80	3
3	$ACBD$	40	2	**15**	$CABD$	70	3
4	$ACDB$	25	2	**16**	$CADB$	55	3
5	$ADBC$	30	3	**17**	$CDBA$	65	3
6	$ADCB$	30	3	**18**	$CDAB$	85	4
7	$BACD$	20	1	**19**	$DACB$	60	4
8	$BADC$	25	2	**20**	$DABC$	60	4
9	$BCAD$	50	2	**21**	$DCAB$	90	5
10	$BCDA$	80	3	**22**	$DCBA$	70	4
11	$BDAC$	55	3	**23**	$DBAC$	40	3
12	$BDCA$	85	4	**24**	$DBCA$	70	4

From Table 2 we observe that a good order for testing the classes with a minimum stubbing effort of 20 is $BACD$. If the application classes are integrated in the order $BACD$, a single stub is required for class D.

While in case of the example from Fig. 2 the permutation with the minimum stubbing cost is the one with the minimum number of stubs, this is not always the case. For example, the permutation $ADBC$ has a stubbing cost of 30 and requires 3 stubs. Permutation $ACBD$ on the other hand has a higher cost, 40, but requires only 2 stubs. In our approach, even if the number of stubs is lower, $ACBD$ is a worse integration order than $ADBC$ because it has a higher cost.

Considering that the application classes from the system from Fig. 2 are numbered as $C_1 = A, C_2 = B, C_3 = C, C_4 = D$, the solution encoded by the GA is a permutation of $\{1, 2, 3, 4\}$ representing a possible integration ordering of the application classes. After applying the GA algorithm with the parameter setting described in Sect. 4.1, the obtained solution is $\{2, 1, 3, 4\}$ corresponding to the class ordering $BACD$ which is shown in Table 2 to be the order which implies the minimum stubbing cost of 20 and the minimum number of stubs of 1.

4.3 Case Studies

For our experiments, we have selected four synthetic case studies and six software systems often used in the CITO literature. In order to mitigate some threats to external validity issues, in our experiments we have chosen software systems of various size, complexity and domain (as shown in Table 3). Even if an industrial software system would probably have more classes, for those systems integration testing would be performed on component-level, instead of class-level. Integration of classes from one component should be done on the class-level [1].

A description of the case studies is given in Table 3, where the second column depicts the number of classes and the third column presents the number of dependencies between the application classes. For each case study, the existing number of cycles between the application classes is given in the fourth column. The number of cycles indicates the complexity of the stubbing process, since a larger number of cycles leads to a larger number of stubs needed in the integration process.

The first synthetic example consists of 8 classes, and its ORD is presented in Fig. 3. This case study was considered for evaluation in several papers approaching the CITO problem [1,3,15,25,26].

Our second synthetic example consists of 8-classes as well, and its Object Relation Diagram is depicted in Fig. 4. This example was previously used in the CITO literature for weighted stubs [10].

The third synthetic example consists of 10 classes and was used in [12]. The ORD of this case study is presented on Fig. 2 from [12] and it contains the same three types of dependency relations that we have considered.

The fourth system that we used as a case study is a real system, denoted by SMDS, which is a Telecommunication Switching System [25]. This system was previously used in the CITO literature by [9,10,25]. We have built the labelled Object-Relation Diagram of SMDS from [25], where both the UML diagram of the system and its corresponding unlabelled ORD is given. The UML diagram contains 37 classes, while the ORD has 38 classes. Since we do not know the labels for the edges of the extra class, our case study has only 37 classes. However, this

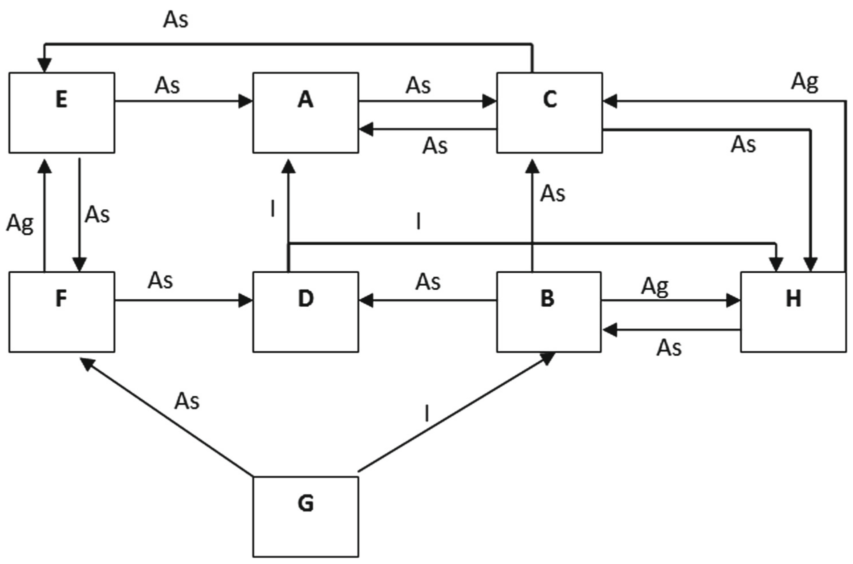


Fig. 3. ORD for the first simple 8-class example [5].

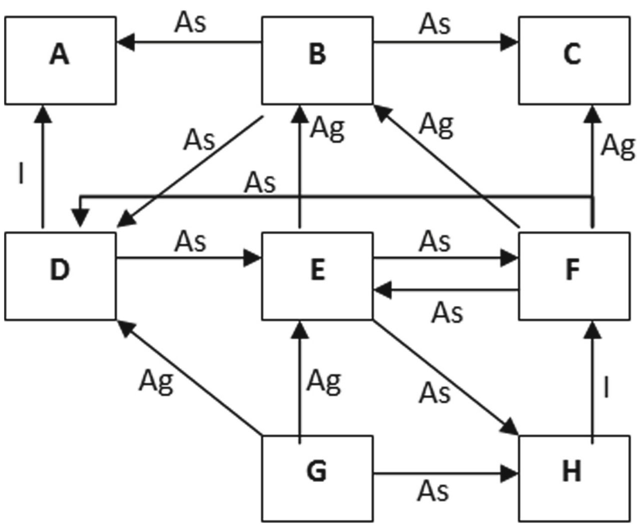


Fig. 4. ORD for the second simple 8-class example [5].

difference in the number of classes does not influence the results, because the extra class has only outgoing edges, it does not participate in any cycles.

The next five case studies used in our experiments are taken from the Briand benchmark [11] and were used in different papers from the CITO literature [1,18, 27]. The systems from this benchmark are the following: ATM (automated teller machine simulation), Ant (a Java-based build tool for maintaining, updating and regenerating related programs and files according to their dependencies), SPM (Security Patrol Monitoring), BCEL (Byte Code Engineering Library, a tool for manipulating binary Java files) and DNS (a Java implementation of the Domain Naming System). For all these systems, we have used the Object Relation Diagram from the Appendix provided by Briand et al. [11].

The last case study is a synthetic example, called Elevator, used in [18]. It consists of 12 classes and its ORD is presented on Fig. 2 from [18].

Table 3. Description of the case studies [5].

Case study	# of classes	# of dependencies	# of cycles
8-class first example [25]	8	17	11
8-class second example [10]	8	16	7
10-class example [12]	10	18	10
SMDS [25]	37	72	35
ATM [1]	21	67	30
Ant [1]	25	83	654
SPM [1]	19	72	1178
BCEL [1]	45	294	416091
DNS [1]	61	276	16
Elevator [18]	12	27	23

4.4 Results

We have applied our GA approach for the case studies presented in Table 3. Results for the equal weighting scheme are presented in Table 4, while results for the differential weighting schemes DW1-DW4 from Table 1 are presented in Table 5. Due to some randomness in the execution, the GA was run 20 times for every weighting scheme. We found that, in every case, the number of stubs reported is the same, except for the Ant case study with equal weighting. For this case study we have run our GA 100 times out of which in 85 cases it reported 9 stubs, while in the other 15 cases it reported 10 stubs. We have to mention that even if our GA obtains the same number of stubs for different runs, the class ordering which produces this number of stubs, as well as the corresponding sequence of stubs, may be slightly different.

For the equal weighting scheme, the class ordering, the number of stubs and the minimum stubbing cost provided by the proposed GA is shown in Table 4. If multiple permutations of classes providing the same number of stubs were obtained by the GA, we illustrate in Table 4 one of them. For the Ant case study, we give the permutation that produces the most frequent number of stubs, which is 9. In Table 5 the stubbing cost and the number of weighted stubs is presented for every case study and differential weighting scheme. When the same stubbing cost and number of stubs is reported for different weighting schemes, they are presented in one single row.

Table 4. Stubs obtained by our GA approach for the case studies for equal weighting.

Case study	Minimum cost	# of stubs	Class order (permutation)
8-class first example	4	4	A, H, D, E, F, C, B, G
8-class second example	2	2	C, A, D, B, F, H, E, G
10-class example	6	6	C2, C4, C3, S1, Rec, S0, S2, C0, S3, C1
SMDS	20	20	18, 37, 20, 6, 17, 10, 29, 14, 13, 22, 16, 25, 24, 11, 33, 32, 2, 7, 23, 8, 34, 12, 26, 1, 9, 27, 5, 19, 28, 3, 15, 21, 36, 35, 31, 30, 4
ATM	7	7	21, 16, 20, 7, 17, 6, 5, 1, 4, 2, 9, 3, 8, 11, 10, 19, 15, 13, 18, 14, 12
Ant	9 or 10	9 or 10	4, 9, 17, 12, 15, 3, 16, 19, 21, 1, 23, 22, 10, 24, 20, 5, 2, 18, 6, 11, 8, 25, 7, 14, 13
SPM	16	16	3, 9, 15, 18, 11, 17, 16, 6, 1, 14, 2, 10, 12, 13, 8, 7, 5, 19, 4
BCEL	58	58	4, 5, 7, 9, 11, 12, 13, 14, 15, 16, 19, 21, 22, 25, 26, 28, 29, 30, 31, 32, 33, 34, 35, 39, 40, 41, 25, 29, 32, 34, 37, 38, 39, 40
DNS	6	6	1, 45, 36, 6, 8, 13, 42, 20, 10, 31, 15, 18, 28, 17, 41, 14, 33, 16, 5, 2, 19, 7, 9, 11, 40, 27, 12, 25, 32, 34, 43, 21, 37, 4, 22, 39, 35, 44, 26, 30, 29, 38, 24, 3, 23
Elevator	5	5	8, 10, 4, 1, 9, 5, 6, 2, 3, 7, 0, 11

5 Discussion

In this section we provide an analysis of our GA approach for both *equal* and *differential* weighting schemes, then a comparison to similar related work from the CITO literature will be conducted.

5.1 Analysis of Out Approach

Analyzing the results from Table 5 we can see that the number of weighted stubs is the same for all four sets of weights in case of seven case studies: 8-class first example, 8-class second example, 10-class example, ATM, SPM, DNS

Table 5. Results obtained by the GA using different weights for the relationships between the classes.

Case study	Differential weighting scheme(s)	Stubbing cost	# of weighted stubs
8-class first example	**DW1, DW2, DW3**	20	**4**
	DW4	80	4
8-class second example	**DW1, DW2, DW3**	10	**2**
	DW4	40	2
10-class example	**DW1, DW2, DW3**	30	**6**
	DW4	120	6
SMDS	**DW1, DW2, DW3**	235	**20**
	DW4	400	20
Ant	DW1, DW3	50	10
	DW2	60	**9**
	DW4	180	9
ATM	**DW1, DW2, DW3**	35	**7**
	DW4	140	7
SPM	**DW1, DW2, DW3**	80	**16**
	DW4	320	16
BCEL	**DW1**	290	**58**
	DW2, DW3	300	60
	DW4	1200	60
DNS	**DW1, DW2, DW3**	30	**6**
	DW4	120	6
Elevator	**DW1, DW2, DW3**	25	**5**
	DW4	100	5

and Elevator. Since the association relation has the smallest weight in every set of weights (even if in DW1 and DW4 there is another relation with the same weight) this means that the optimal class integration test order can be achieved by stubbing only association relations. This can also be seen by the fact that for these case studies the stubbing cost (given in the third column of Table 5) is equal to the weight of the association relation multiplied by the number of weighted stubs (given in the last column of Table 5). Looking at the results returned by the GA we observed that in case of DW1 and DW4, where there is another relation with the same weight as the association, it is possible that the best solution requires the stubbing of an inheritance (for DW1) or aggregation (for DW4) relation. However, the costs from DW2 and DW3 show that there exist solutions where only association relations are stubbed.

Interestingly, in case of the SMDS system, all four sets of weights return the same number of stubs, 20, but from the associated Stubbing cost we can see that for this case study we cannot find an integration order where only association relations are broken. In the first three sets of weights, the weight of an association relation is 5, so the cost of breaking 20 association edges would be 100, but the cost reported in Table 5 is 235. Checking the exact integration orders returned by the GA, we observed that in all situations 11 associations and 9 aggregations were stubbed, and since in the first three weighting schemes these relations have the same weights (5 and 20 respectively) the stubbing cost was the same. The reason why we need these 9 aggregation relations stubbed is the presence of 9 two-class circles where both edges from the circle have the aggregation label.

In case of the Ant system, the sets of weights for DW1 and DW3 return a solution where 10 specific stubs are required, while the other two weighting schemes return solutions that require 9 stubs. The stubbing cost from DW3 shows that there is a solution where only association relations are stubbed, but it requires 10 stubs. If we want to reduce the number of stubs, we have to stub an aggregation relation as well.

For the BCEL system, the DW1 scheme returns a solution where 58 stubs are required, while all other schemes return solutions requiring 60 stubs. This means that we can find again a solution where only association relation are stubbed, but it requires two extra stubs compared to the solution from DW1. In case of DW1, out of the 58 specific stubs that have to be created for the given class integration order, four inheritance and 54 association relations have to be stubbed.

Comparing the results from Tables 4 and 5 we can see that the number of required stubs for equal weighting is always equal to the minimum number of stubs from the differential weighting scheme. This is not surprising, since equal weighting does not differentiate between types of relations, all dependencies have a weight of 1, shown by the fact that in Table 4 the minimum cost is always equal to the number of stubs.

The results from Table 5 are very different from the ones reported by Malloy and Kraft for similar weight settings. While they reported significant differences in the number of required stubs depending on whether inheritance had a large weight or not, in our experiments for most case studies there was no difference in the number of stubs and even for those two case studies where there was a difference it was really small (1 and 2 stubs). Unfortunately, the case studies used by Malloy and Kraft are not openly available on the internet, so we could not run our algorithm on them.

In order to try and explain this difference, we checked whether the systems used by Malloy and Kraft in [13,14] had a larger percentage of inheritance relations than our systems, but actually the opposite is true. In our case studies, on average 12% of the relations are inheritance (minimum is 0 for Elevator, maximum is 22% for the 10-class example). If we consider only the six real life case studies, which are also the systems with a larger number of classes, the average is 11%. In case of the systems used by Malloy and Kraft (both papers use the same systems) on average only 4% of the dependencies are inheritance (minimum is 0, maximum is 8%).

In general, Malloy and Kraft used larger systems than our case studies. However, if we look only at the results of their case studies with 43–50 classes (4 out of the 7 case studies have a number of classes in this interval) for two of them there still is a significant difference in the number of required stubs (approximately 25 stubs) between the two main settings (do not break inheritance at all or break anything). For the other two systems the difference is really small (0 and 1 stubs).

In conclusion the reason for the difference between the results reported in [13,14] and the findings of our experiments might be some difference between the systems that cannot be found by looking at the size of the system or the percentage of inheritance relations. We mention that most of our systems are written in Java while the systems used by Malloy and Kraft are written in C++. However, we are not sure whether this difference is relevant or not, but we intend to continue our investigation using other case studies of different sizes and from different programming languages.

From Tables 4 and 5 we can observe that for each case study, for all the differential weighting schemes DW1-DW4, the GA provided the same number of stubs for all the 20 runs of the algorithm. Thus, if we are looking at the obtained number of stubs, under the parameter setting from Sect. 4.1, our GA with stochastic acceptance has a deterministic behavior for differential weighting for all case studies. For the equal weighting scheme for the Ant case study we had two different solutions (but the solution with the less number of stubs is the most frequently reported one). This suggests that the differential weighting might contribute to making the GA deterministic.

5.2 Comparison to Related Work

In the following we provide a comparison of the results obtained by the proposed GA approach to the results reported in the literature for the case studies considered for evaluation. The comparison is depicted in Table 6. For the simple case studies we have also computed the optimal number of stubs using a brute force approach. For the larger software systems, the brute force method is not feasible, due to its exponential time complexity. The best result obtained using a non-brute force approach (i.e., the one which reports the minimum number of weighted stubs) is highlighted. For each result we indicate the paper where the result was taken from.

For the brute force approach we implemented a parallel algorithm based on Heap's algorithm for generating permutations and we run the experiment on a server machine with 16 cores. The running times for the larger systems are over 48 h while the proposed GA approach finds the optimal ordering in less than 3 min/run for every system (the genetic algorithm is not parallel, it uses only a single core). For the two smaller systems the required time is less than 30 s.

For the 8-class examples, we found in the literature results reported considering the *differential weighting* scheme and the same weights as in DW2 [3]. For the 10-class example and the Elevator case studies, we have found results for weighted approaches, but with weights considered differently than in our approach. Unlike for the simple case studies, for the case studies from the Briand benchmark and SMDS there are no results reported in the literature considering weighted stubs as in our approach (i.e. assigning different weights for specific types of relationships between the application classes). For these systems, only the results obtained for *generic* and *specific* stubs are available. That is why, for

Table 6. Comparison to related work considering *weighted* and *specific* stubs [5].

#	Case study	Weighting scheme	Approach	# of stubs
1	8-class first example	Differential weighting (DW2)	**Our GA solution**	**4**
			Brute force	4
			Tai et al. [3]	5
			Le Traon et al. [3]	5
			Le Traon et al. [3]	6
			Le Traon et al. [3]	4
			Briand et al. [3]	4
			Malloy et al. [3]	6
			Abdurazik et al. [3]	4
			AICTO [12]	4
2	8-class second example	Differential weighting (DW2)	**Our GA solution**	**2**
			Brute force	2
			Kung et al. [10]	4
			Tai and Daniels [10]	2
			Hanh et al. - Triskell strategy[10]	2
			Hanh et al. - Genetic algorithm [10]	3
3	10-class example	Differential weighting	**Our GA solution**	**6**
			Brute force	6
			Tai [12]	7
			Briand [12]	8
			Briand [12]	7
			AICTO [12]	6
4	SMDS	Equal Weighting	**Our GA solution**	**20**
			Le Traon et al. – Optimal [9]	20
			Le Traon et al. – RC [9]	25–48
			Le Traon et al. – MC [9]	26
			Le Traon et al. – RT [9]	27-47
			Le Traon et al. – MT [9]	28–38
			Buordoncle algorithm [9]	23
5	Ant	Equal weighting	**Our GA solution**	**9 or 10**
			Briand et al. [1]	11
			Tai and Daniels [1]	28
			Le Traon et al. [1]	19
6	ATM	Equal weighting	**Our GA solution**	**7**
			Briand et al. [1]	7
			Tai and Daniels [1]	8
			Le Traon et al. [1]	7
7	SPM	Equal weighting	**Our GA solution**	**16**
			Briand et al. [1]	17
			Tai and Daniels [1]	20
			Le Traon et al. [1]	27
8	BCEL	Equal weighting	**Our GA solution**	**58**
			Briand et al. [1]	70
			Tai and Daniels [1]	128
			Le Traon et al. [1]	67
9	DNS	Equal weighting	**Our GA solution**	**6**
			Briand et al. [1]	6
			Tai and Daniels [1]	27
			Le Traon et al. [1]	10
10	Elevator	Differential weighting	**Our GA solution**	**5**
			Brute force	5
			Zhang [18]	5

the case studies from the Briand benchmark and the SMDS system we applied our GA with stochastic acceptance under the *equal weighting* scheme which is equivalent to obtaining the *specific* stubs.

The first line from Table 6 shows the results for the first 8-class example from Fig. 3. Bansal et al. [3] report the results obtained by several approaches from the literature on this case study. The approach from Le Traon et al. provides multiple solutions with different number of stubs and we included all of them in the table. Results for this case study are reported in [12] as well. The second line of Table 6 contains the comparison of the results for the second 8-class example from Fig. 4. This case study was previously used by Le Hanh et al. [10], where the result of several approaches are reported for it.

The third line contains the results for the 10-class case study, which was used in [12], where the results of Tai's and Briand's graph-based algorithms are also reported for it (for Briand two results are reported). All three approaches (Tai, Briand and AICTO itself) break only association relations so we compare these results with the results of differential weighting.

For our fourth case study, the SMDS software, we found in the literature the results reported by Le Traon et al. in [9] for various approaches for the CITO problem considering only non-weighted specific stubs (i.e. the *equal weighting* scheme from our approach).

The next lines from Table 6 depict the comparison of the results obtained using our GA with those provided by Briand et al. [1] on Ant, ATM, SPM, BCEL and DNS systems. The last row from Table 6 shows the results for the Elevator case study, which was used only in [18].

Table 6 shows that the results provided by our GA approach are better than or at least equal to the results reported by approaches existing in the literature considering the case studies used in our experiments. In **11** cases, the number of

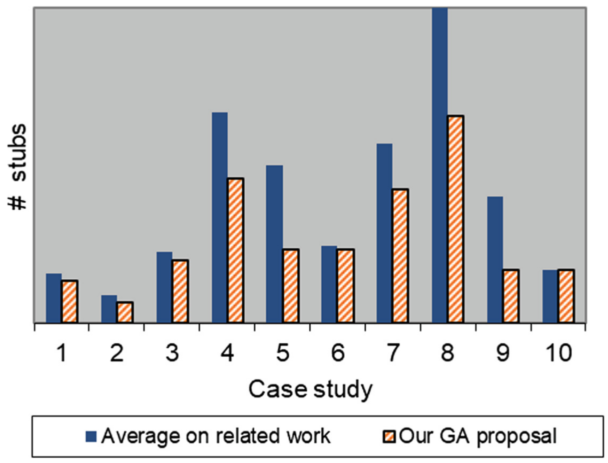


Fig. 5. GA performance compared to the average performance of the related work on the considered data sets.

weighted stubs obtained is the same as the one from the related work, while in **26** situations a smaller number of weighted stubs was obtained by our approach. The comparison to the related work is graphically illustrated in Fig. 5. For each case study we represent the average number of weighted stubs reported in the literature and through the dashed bars the number of weighted stubs reported by our GA solution.

6 Conclusions and Future Work

We extended in this paper our previous approach from [5] which focused on the problem of *class integration test ordering* and introduced a genetic algorithm with stochastic acceptance for determining an integration test order strategy for object-oriented systems. The goal was to identify, based on a static analysis of object-oriented software systems, the test order requiring a minimum stubbing effort. We considered a weighted cost for creating the specific stubs needed for testing. Ten case studies were considered in our experimental evaluation, both synthetic examples and systems used in the literature for the CITO problem. The results obtained using our approach outperformed those of existing similar work.

An analysis of the GA results considering different weights for the relationships between the application classes was performed with the aim of investigating the importance of the relationships between the classes in the stubbing process. The result of our analysis showed that, at least for most of the considered systems, changing the weights assigned to different relations does not influence the number of required stubs, since most stubbed relations are associations anyway. For the case studies where there was a difference between the number of stubs, this was very small.

In the future, we aim to apply the GA proposed in this paper for large scale software systems in order to test its scalability. For a better handling of systems with a large number of application classes, a parallel implementation of the GA will be further developed. We will also investigate new dependencies between application classes (such as *direct or indirect coupling*) for computing the *stubbing effort*, as well as assigning weights not only to the dependencies in the ORD, but also to the application classes. Another direction which we will investigate in the future is that of learning (possibly through supervision or reinforcement) appropriate values for weighting the dependencies between the application classes from the software systems.

References

1. Briand, L.C., Labiche, Y., Wang, Y.: Revisiting strategies for ordering class integration testing in the presence of dependency cycles. Technical report TR SCE-01-02, Carleton University (2002)

2. Assunção, W.K.G., Colanzi, T.E., Pozo, A.T.R., Vergilio, S.R.: Establishing integration test orders of classes with several coupling measures. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO 2011. ACM, New York, pp. 1867–1874 (2011)
3. Bansal, P., Sabharwal, S., Sidhu, P.: An investigation of strategies for finding test order during integration testing of object oriented applications. In: Proceedings of International Conference on Methods and Models in Computer Science, pp. 1–8 (2009)
4. Briand, L.C., Feng, J., Labiche, Y.: Using genetic algorithms and coupling measures to devise optimal integration test orders. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, SEKE 2002, pp. 43–50. ACM, New York (2002)
5. Czibula, I.G., Czibula, G., Marian, Z.: An improved approach for class test ordering optimization using genetic algorithms. In: Proceedings of the International Conference on Software Technologies, Madrid, Spain, pp. 23–37 (2017)
6. Melton, H., Tempero, E.: An empirical study of cycles among classes in Java. Empir. Softw. Eng. **12**, 389–415 (2007)
7. Kung, D., Gao, J., Hsia, P., Toyoshima, Y., Chen, C.: A test strategy for object-oriented programs. In: Proceedings of the Nineteenth Annual International Computer Software and Applications Conference, COMPSAC 1995, pp. 239–244 (1995)
8. Tai, K.C., Daniels, F.J.: Test order for inter-class integration testing of object-oriented software. In: Proceedings of the 21st International Computer Software and Applications Conference, COMPSAC 1997, pp. 602–607. IEEE Computer Society, Washington, DC (1997)
9. Le Traon, Y., Jéron, T., Jezequel, J.M., Morel, P.: Efficient object-oriented integration and regression testing. IEEE Trans. Reliab. **49**, 12–25 (2000)
10. Le Hanh, V., Akif, K., Le Traon, Y., Jézéque, J.-M.: Selecting an efficient OO integration testing strategy: an experimental comparison of actual strategies. In: Knudsen, J.L. (ed.) ECOOP 2001. LNCS, vol. 2072, pp. 381–401. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45337-7_20
11. Briand, L.C., Feng, J., Labiche, Y.: Experimenting with genetic algorithms and coupling measures to devise optimal integration test orders. Technical report TR SCE-02-03, Carleton University (2002)
12. Mao, C., Lu, Y.: AICTO: an improved algorithm for planning inter-class test order. In: Proceedings of the 2005 International Conference on Computer and Information Technology, pp. 927–931 (2005)
13. Malloy, B.A., Clarke, P.J., Lloyd, E.L.: A parameterized cost model to order classes for class-based testing of C++ applications. In: 14th International Symposium on Software Reliability Engineering, ISSRE 2003, pp. 353–364 (2003)
14. Kraft, N.A., Lloyd, E.L., Malloy, B.A., Clarke, P.J.: The implementation of an extensible system for comparison and visualization of class ordering methodologies. J. Syst. Softw. **79**, 1092–1109 (2006)
15. Abdurazik, A., Offutt, J.: Using coupling-based weights for the class integration and test order problem. Comput. J. **52**(5), 557–570 (2009)
16. Guizzo, G., Fritsche, G.M., Vergilio, S.R., Pozo, A.T.R.: A hyper-heuristic for the multi-objective integration and test order problem. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO 2015, pp. 1343–1350. ACM, New York (2015)

17. Mariani, T., Guizzo, G., Vergilio, S.R., Pozo, A.T.: Grammatical evolution for the multi-objective integration and test order problem. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO 2016, pp. 1069–1076. ACM, New York (2016)
18. Zhang, M., Jiang, S., Zhang, Y., Wang, X., Yu, Q.: A multi-level feedback approach for the class integration and test order problem. J. Syst. Softw. **133**, 54–67 (2017)
19. Abdurazik, A., Offutt, J.: Coupling-based class integration and test order. In: Proceedings of the 2006 International Workshop on Automation of Software Test, pp. 50–56 (2006)
20. Le Traon, Y., Jéron, T., Jézéquel, J.M., Morel, P.: Efficient OO integration and regression testing. IEEE Trans. Reliab. **49**, 12–25 (2000)
21. Whitley, D.: An overview of evolutionary algorithms: practical issues and common pitfalls. Inf. Softw. Technol. **43**, 817–831 (2001)
22. Melanie, M.: An Introduction to Genetic Algorithms. The MIT Press, Cambridge (1999)
23. Lipowski, A., Lipowska, D.: Roulette-wheel selection via stochastic acceptance. Phys. A: Stat. Mech. Appl. **391**, 2193–2196 (2012)
24. Whitley, D., Yoo, N.-W.: Modeling simple genetic algorithms for permutation problems. In: Foundations of Genetic Algorithms, pp. 163–184. Morgan Kaufmann (1995)
25. Hewett, R., Kijsanayothin, P.: Automated test order generation for software component integration testing. In: Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, ASE 2009, pp. 211–220. IEEE Computer Society, Washington, DC (2009)
26. Borner, L., Paech, B.: Integration test order strategies to consider test focus and simulation effort. In: Proceedings of the First International Conference on Advances in System Testing and Validation Lifecycle, pp. 80–85 (2009)
27. da Veiga Cabral, R., Pozo, A., Vergilio, S.R.: A pareto ant colony algorithm applied to the class integration and test order problem. In: Petrenko, A., Simão, A., Maldonado, J.C. (eds.) ICTSS 2010. LNCS, vol. 6435, pp. 16–29. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16573-3_3

Application of Fuzzy Logic to Assess the Quality of BPMN Models

Fadwa Yahya[1](✉), Khouloud Boukadi[1], Hanêne Ben-Abdallah[2], and Zakaria Maamar[3]

[1] Sfax University, Sfax, Tunisia
{inaya.yahya,khouloud.boukadi}@fsegs.rnu.tn
[2] King Abdulaziz University, Jeddah, Kingdom of Saudi Arabia
hbenabdallah@kau.edu.sa
[3] Zayed University, Dubai, UAE
zakaria.maamar@zu.ac.ae

Abstract. Modeling is the first stage in a Business Process's (BP) lifecycle. A high-quality BP model is vital to the successful implementation, execution, and monitoring stages. Different works have evaluated BP models from a quality perspective. These works either used formal verification or a set of quality metrics. This paper adopts quality metric and targets models represented in Business Process Modeling and Notation (BPMN). It proposes an approach based on fuzzy logic along with a tool system developed under eclipse framework. The preliminary experimental evaluation of the proposed system shows encouraging results.

Keywords: Business Process · BPMN · Model quality
Quality metrics · Fuzzy logic

1 Introduction

A Business Process (BP) model covers different dimensions of an enterprise such as functional, organizational, behavioral, and informational [4]. Integrating all these dimensions into one *high-quality* model is vital to the survivability of any enterprise [17,22]. Indeed, such a model will support activities such as implementation, deployment, execution, and continuous improvement in short, the BP lifecycle [31]. A high-quality BP model will also guarantee its acceptance by end-users and thus prevents common BP problems like model reality divide where the modeled and executed processes are not aligned [26].

In the literature, BP model quality assessment has been dealt with using two main approaches: application of formal verification techniques [15,30], or evaluation of a set of quality metrics calculated on the BP model [14,17,22]. Formal techniques are known for their complexity and thus, are unpopular. In addition, they do not provide any qualitative analysis of the model in terms of complexity, comprehensibility, and modifiability.

E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 188–209, 2018.
https://doi.org/10.1007/978-3-319-93641-3_9

Adopting a qualitative assessment of BP models, researchers proposed to calculate a set of metrics either on static (e.g., [14,17,22]), or simulated BP model (e.g., [8]). In these works, several quality metrics are used either to assess certain quality characteristics of the BP model itself (complexity, maintainability, integrity, etc. [11,14,20]) or to predict the BP performance (the mutual impact between the BP and its underlying information system [8]). The main challenges in metric-based assessment are: what are the quality characteristics of a BP model? How to relate the metrics to quality characteristics? And how to interpret the values of the metrics?

The lack of consensus over the quality characteristics of BP models is making the assessment of these models a challenge. Several researchers explored the similarities between processes and software products to adopt these products' quality characteristics. In particular, they adopted the eight quality characteristics defined in the ISO/IEC 25010 [9] standard quality model, (e.g., [20,24]). Because ISO/IEC 25010 quality model does not define any technique for evaluating the characteristics, different studies recommend metrics for assessing the quality of BP models (e.g., [3,20,24,27,28]). In addition, based on the quality metrics, some researchers proposed the development of an automated framework to evaluate BP model quality, e.g., [14,22,25]. The common barrier hindering the development of such a framework is the lack of a consensus about threshold values of the quality metrics, which are required to interpret/evaluate a BP model's quality [17,22].

This paper, which is a revised and extended version of our paper presented at the 12th International Conference on Software Technologies (ICSOFT 2017) [32], addresses the lack of consensus of quality metrics along with their assessment through a fuzzy logic-based approach for evaluating the quality of BP models with an emphasis on comprehensibility and modifiability characteristics. The choice of these characteristics is justified by their importance to guarantee that a BP model can be easily implemented, deployed, and executed. These characteristics also are important when dealing with the continuous improvement of a BP.

The herein proposed approach consists of two phases: threshold determination and fuzzy logic application. The first phase applies data mining, specifically decision trees, to determine approximate thresholds for each quality metric. These thresholds will be used for interpreting the comprehensibility or modifiability levels of BP models in BPMN [10]. To this end, we use a BP repository, called "SOA-based Business Process Database"[1], developed by our Mir@cl laboratory team. This repository contains 1000 BPs of organizations operating in different sectors. The second phase uses the approximate thresholds identified in the first phase along with fuzzy logic [33] to assess the quality of a BPMN model. The use of fuzzy logic aims at dealing with the approximate and imprecise nature of the obtained thresholds. Indeed, according to Zadeh, fuzzy logic operates perfectly in an environment of "imperfect information" [34].

[1] https://sites.google.com/site/bposcteam2015/ressources.

The proposed approach is implemented in a system that allows the qualitative assessment of BPMN models in terms of comprehensibility and modifiability. To prove the performance of the proposed system, we conducted two types of experiments. The first involves students from the faculty of Economics and management of Sfax while the second is done through the proposed system. These preliminary experimental evaluations of the proposed system show encouraging results.

This paper has 3 contributions: (i) identification of approximate thresholds for the different quality metrics to be used for assessing the quality of BP models in terms of comprehensibility and modifiability; (ii) management of the approximate and imprecise nature of the identified thresholds using fuzzy logic; and (iii) development of a system that supports the proposed approach.

The remainder of this paper is organized as follows: Sect. 2 summarizes existing works on the adoption of quality metrics and the definition of their thresholds for the assessment of the comprehensibility and modifiability of BP models. Section 3 presents the proposed approach mining metrics' thresholds. Section 4 shows how we use fuzzy logic to support the approximate and imprecise nature of the defined thresholds. Section 5 illustrates the developed system of BP model quality assessment and evaluates it through two types of experiments. Finally, Sect. 6 summarizes the paper and gives some directions for future work.

2 Related Work

We first introduce the quality metrics used for assessing BP models quality. Then, we discuss works on BP model quality assessment.

2.1 Quality Metrics

Different research initiatives adopt quality metrics from software engineering to assess the quality of BP models as BPs are a kind of software systems [2,5,19,24].

To identify the necessary metrics, we conducted a literature review on existing quality metrics for assessing the comprehensibility and modifiability levels of BP models. To shortlist the relevant metrics, we raised the following questions:

1. Is the metric validated either theoretically or empirically?
2. Is there a technique for calculating the metric?
3. Is it possible to calculate the metric for a BP in BPMN?
4. Is the metric used to evaluate the comprehensibility and/or modifiability of BP models?

At the end of this study, only a few number metrics were selected. The metrics were eliminated essentially by the first question; indeed, several metrics are adopted from the software engineering domain but they are not validated in the BP domain theoretically nor empirically [16]. In addition, some metrics were excluded because of the 3rd question, i.e., they are not adopted for BPMN [20]. The retained metrics are listed below and detailed in [2,12,28,29]:

- Control Flow Complexity (CFC) measures the complexity introduced by XOR, OR, and AND split constructs.

  $$CFC(p) = \sum_{a \in P \wedge a \in XOR-Split} CFC_{XOR}(a) + \sum_{a \in P \wedge a \in OR-Split} CFC_{OR}(a) + \sum_{a \in P \wedge a \in AND-Split} CFC_{AND}(a) \tag{1}$$

  where:
  - $CFC_{XOR}(a) = n$;
  - $CFC_{OR}(a) = 2^n - 1$;
  - $CFC_{AND}(a) = 1$;
  - n = number of outgoing arcs.
- Halstead-based Process Complexity (HPC) estimates the length N, volume V, and difficulty D of a process as follows:

  $$N = n_1 * \log_2(n_1) + n_2 * \log_2(n_2) \tag{2}$$

  $$V = (N_1 + N_2) * \log_2(n_1 + n_2) \tag{3}$$

  $$D = (\frac{n_1}{2}) * (\frac{N_2}{n_2}) \tag{4}$$

  where:
  - n_1 is the number of activities, splits and joins, and control-flow elements of a BP.
  - n_2 is the number of data variables manipulated by the BP and its activities.
  - N_1 and N_2 are respectively the total number of elements and data occurrences.
- Interface Complexity (IC) measures the complexity of a process as follows:

  $$IC = length * (NbOfInputs * NbOfOutputs)^2 \tag{5}$$

- Number of Activities (NOA) measures the number of activities (task and sub-process) of a BP.
- Number of Activities, Joins and Splits (NOAJS) measures number of activities, joins, and splits of a BP.
- Coefficient of Network Complexity (CNC) is the ratio of the total number of arcs in a process model to its total number of nodes.
- Cross Connectivity (CC) expresses the sum of the connectivity between all pairs of nodes, relative to the theoretical maximum number of paths between all nodes.
- Coupling metric (CP) calculates the coupling degree of a process. This coupling degree depends on the complexity of connections between the tasks and the type of these connections (i.e., AND, OR, XOR).
- Density (D) is the ratio of the total number of arcs to the maximum number of arcs.

Table 1 shows the usability of these metrics for measuring either comprehensibility and/or modifiability as defined in the literature.

Table 1. Identified quality metrics for assessing comprehensibility and modifiability [32].

Quality metrics	Comprehensibility	Modifiability
CFC	⋆	⋆
HPC		⋆
IC	⋆	
NOA	⋆	⋆
NOAJS	⋆	⋆
CNC	⋆	⋆
CC	⋆	
CP	⋆	⋆
D	⋆	⋆

⋆: The metric is used to assess the comprehensibility or the modifiability.

2.2 Business Process Evaluation

Despite the importance of BP models to enterprises, there is a serious lack of an effective approach and system for assessing BP models quality [20,24]. Our literature review revealed that ISO standards for quality assessment and quality metrics are the basis of the different assessment tentatives. However, the lack of thresholds for the defined quality metrics to be used during BP models quality assessment remains a concern.

Makni et al. [11] propose a tool for evaluating the quality of BP models using existing complexity, coupling, and cohesion quality metrics. However, the authors did not focus on the identification of thresholds as the proposed tools ensure the evaluation based on thresholds introduced by the user for the different metrics.

Sánchez-González et al. use the Bender method [1] to identify thresholds for some quality metrics [22]. The Bender method allows quantitative risk assessment in epidemiological studies based on the logistic regression model. It has two major limitations: (i) logistic regression model requires a binary variable and (ii) necessity to arbitrarily define $P0$ probability, which is used to calculate the Value of an Acceptable Risk Level (VARL). The Bender method was also used in [25] to define threshold for the CFC metric. In [23] the same authors conduct an experiment to determine threshold values for gateway complexity metrics to be used for the evaluation of the understandability and modifiability of BP models. The authors also propose a Gateway Complexity Indicator (GCI) defined based on the identified threshold values for the selected gateway complexity measures.

Mendling et al. propose an approach for predicting errors in BP models [14]. The approach uses a set of quality metrics included in the literature for evaluating the quality of BP models. The authors use logistic regression [1] and ROC curves [7] to determine thresholds for the concerned metrics.

Sadowska proposed a meta-model for assessing the quality of BPMN 2.0 process models [20]. This meta-model is built upon the ISO/IEC 25010 standard [9]. To evaluate the different quality characteristics, the author uses a set of quality metrics like CFC and density. In addition, she used a BP repository of 57 BPs in BPMN along with K-means to classify the possible values of quality metrics into 4 clusters. Based on the used quality metrics and the defined clusters the author proposed a system that supports the evaluation of BP models quality.

Our literature review reveals the lack of a consensus concerning thresholds values for BP models quality assessment. This is one of the important obstacles hindering the development of an effective system supporting qualitative assessment of BP models.

3 Determination of Quality Metrics Thresholds

We detail our approach for estimating thresholds of evaluating BP model quality in terms of modifiability and comprehensibility. This approach is based on a data mining technique for instance decision tree. Four steps are required: data collection to build a repository of BPs, data preparation to create the learning and test datasets, data mining to build a decision tree, and finally validation to assess the performance of the resulted decision tree.

3.1 Data Collection

As part of the research agenda of our lab, we created "SOA-based Business Process Database" by collecting around 1000 BPs that belong to different institutions, this should guarantee that our approach is generic (e.g., academic institutions, commercial enterprises, healthcare centers, and banks). Furthermore, from each type of organization, we examined different BPs; for example, from academic institutions, we considered, among others, student registration, exam preparation, timetable creation, etc. All these BPs are modeled in BPMN 2.0.

After data collection, we examined the processes in conjunction with design instructors from the IT department of our university (considered as experts). The goal is to classify these processes according to how easy they are to comprehend and modify. To this end, we organized ourselves into four groups. Each group examined 250 processes. Afterward, we conducted a cross-validation process among the different groups. Finally, we organized the BPs of the "SOA-based Business Process Database" into three levels of comprehensibility (easy to understand, moderately difficult to understand, and difficult to understand) and three levels of modifiability (easy to modify, moderately difficult to modify, and difficult to modify).

3.2 Data Preparation

To prepare the data for the next phases, we built two matrices based on the "SOA-based Business Process Database". The first is devoted to comprehensibility data, while the second is dedicated to modifiability data. Each row in a matrix represents a BP in the database and each column represents a quality metric among the identified quality metrics to measure comprehensibility and modifiability (Sect. 2.1). The last column of each matrix represents the aforementioned levels of comprehensibility and modifiability.

We used these matrices to create two sub-databases from each matrix: one for learning known as "training database" and one for testing known as "test database". The "training database" includes 70% of the processes in the database, and the "test database" comprises the rest.

3.3 Data Mining

We used decision trees to extract thresholds for quality metrics from the "SOA-based Business Process Database". For more details about decision trees readers are referred to [18,21].

To create the required decision trees (for comprehensibility and modifiability), we used WEKA system, which is recognized as a landmark system in data mining and machine learning [6]. WEKA supports several algorithms for the construction of decision trees like J48, ADTree, and REPTree. In this work, we first used all of the provided algorithms, and then we have chosen the best one (i.e., the one that have a lower error rate) based on the validation stage (Sect. 3.4).

3.4 Validation

The literature proposes several possible ratios for assessing the quality of a prediction model. We used: precision (6), recall (7), f-measure (8), and global error rate (9). In the following, we discuss J48, ADTree, and REPTree algorithms, selected because of their acceptable values of the used ratios.

$$Precision = \frac{CorrectEntitiesFound}{TotalEntitiesFound} \tag{6}$$

$$Recall = \frac{CorrectEntitiesFound}{TotalCorrectEntities} \tag{7}$$

$$F\text{-}measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{8}$$

$$GlobalErrorRate = 1 - \frac{CorrectEntitiesFound}{TotalEntities} \tag{9}$$

– **Training Database-based Validation**
  First, we calculate these ratios after testing the resulting decision trees (i.e., comprehensibility and modifiability trees) on the training database. Tables 2 and 3 show, respectively, the values of the different ratios for the three algorithms per decision tree. These tables depict that J48 algorithm gives the best values of precision, recall, F-measure, and global error rates for both comprehensibility and modifiability.

Table 2 shows that we achieved very acceptable results with J48, for assessing the comprehenisibility of BP models: the values of precision, recall, and F-measure are 97.3% and the global error rate is 2.7%. Similarly, Table 3 shows that J48 can also be used for assessing the modifiability of BP models as the values of precision, recall, and F-measure are 96.1% while the global error rate is 3.8%. However, to prove the effectiveness of the proposed decision trees, we need to use another database like the "test database".

Table 2. J48 *versus* ADTree *versus* REPTree for comprehensibility decision-tree [32].

	J48	ADTree	REPTree
Precision	0.973	0.961	0.942
Recall	0.973	0.96	0.94
F-measure	0.973	0.96	0.941
Global error rate	0.027	0.04	0.06

Table 3. J48 *versus* ADTree *versus* REPTree for modifiability decision-tree [32].

	J48	ADTree	REPTree
Precision	0.961	0.92	0.925
Recall	0.961	0.92	0.924
F-measure	0.961	0.92	0.924
Global error rate	0.038	0.08	0.075

– **Test Database-based Validation**
To assess the performance of the proposed decision trees and choose the most suitable algorithm among those provided by WEKA, we evaluated the obtained trees using the "test database". At this stage, we apply each decision tree to all BPs of the "test database" to assess the comprehensibility and modifiability levels of each process. This assessment is performed independently of the assessments already done by experts. The goal is to compare the experts' judgments with the obtained assessments trees and hence, to identify the error rate of our decision trees.
Tables 4 and 5 list the values of the four used ratios for evaluating the performance of the proposed comprehensibility and modifiability decision trees. These tables show that we achieved very acceptable results using the "test database" and J48. Indeed, the values of precision are: 96.9% for comprehensibility tree and 94.3% for modifiability tree. The values of recall and f-measure are: 96.7% for comprehensibility and 94% for modifiability. Finally, the global error rate is 3.3% for comprehensibility tree and 6% for modifiability tree.

Table 4. J48 *versus* ADTree *versus* REPTree for comprehensibility decision-tree [32].

	J48	ADTree	REPTree
Precision	0.969	0.971	0.962
Recall	0.967	0.967	0.953
F-measure	0.967	0.968	0.955
Global error rate	0.033	0.033	0.046

Table 5. J48 *versus* ADTree *versus* REPTree for modifiability decision-tree [32].

	J48	ADTree	REPTree
Precision	0.943	0.875	0.92
Recall	0.94	0.87	0.897
F-measure	0.94	0.869	0.899
Global error rate	0.06	0.13	0.103

3.5 Discussions

Decision tree is used to classify the BPs of the "SOA-based Business Process Database" according to their level of comprehensibility (first decision tree) and modifiability (second decision tree). Based on these decision trees, we defined some decision rules along with the thresholds of the different quality metrics for evaluating both comprehensibility and modifiability of a BP model.

1. **Evaluation of the Comprehensibility Level**
   Table 6 presents the identified thresholds and their linguistic interpretations. These interpretations are performed by the members of our research team. However, the identified thresholds remain usually approximate and imprecise as they depend on the experts' judgments during the first phase, "data collection" (Sect. 3.1). In the next Section, we detail the use of fuzzy logic to manage these approximate and imprecise thresholds.
   Table 7 presents an extract of the decision rules that determine the comprehensibility level based on the values of the quality metrics.
2. **Evaluation of the Modifiability Level**
   Table 8 presents the identified thresholds and their linguistic interpretations, which are determined by the members of our research team. Same as the case of comprehensibility, the identified thresholds are approximate and imprecise. To handle this approximate and imprecise nature of thresholds, we use fuzzy logic (Sect. 4).
   Table 9 presents an extract of the decision rules that determine the modifiability level based on the values of the quality metrics.

Table 6. Thresholds for the evaluation of comprehensibility.

Quality metric	Thresholds	Linguistic interpretation
IC	$IC < 12$	Low
	$12 \leq IC < 17$	Moderate
	$17 < IC \leq 59$	High
	$IC > 59$	Very high
CNC	$CNC \leq 1.26$	Low
	$CNC \leq 1.65$	Moderate
	$CNC > 1.65$	High
NOAJS	$NOAJS \leq 39$	Low
	$39 < NOAJS \leq 55$	Moderate
	$NOAJS > 55$	High
CFC	$CFC \leq 3$	Low
	$3 < CFC \leq 9$	Moderate
	$9 < CFC \leq 18$	High
	$CFC > 18$	Very high
NOA	$NOA \leq 26$	Low
	$26 < NOA \leq 45$	Moderate
	$NOA > 45$	High
D	$D \leq 0.043$	Low
	$D > 0.043$	High
CP	$CP \leq 0.031$	Low
	$CP > 0.031$	High

4 Fuzzy Logic for Business Process Quality-Assessment

In this paper, we use fuzzy logic along with the identified thresholds to assess the quality of BP models. Fuzzy logic is known for being appropriate for handling approximate and imprecise values like those for the quality metrics' thresholds. Fuzzy-logic application goes through 3 stages: fuzzification, inference, and defuzzification. The fuzzification is based on a set of membership functions that transform the crisp values of the quality metrics into linguistic values (e.g., low, medium, high) known as fuzzy sets. The inference uses a set of fuzzy decision rules to assess the quality of BP models. Finally, the defuzzification step produces a quantifiable (crisp value) result. In the remainder of this Section, we detail the use of fuzzy logic to assess the quality of BP models.

4.1 Fuzzification

Fuzzification converts crisp values of input variables (i.e., quality metrics) into fuzzy sets (i.e., linguistic values). This conversion is ensured thanks to a set

Table 7. Excerpt of decision rules to assess the level of comprehensibility [32].

	Decision rule
R1	If $IC \leq 12$ Then Easy to understand
R2	If $IC \leq 17$ and $IC > 12$ and $CNC \leq 1.26$ Then Easy to understand
R3	If $IC \leq 17$ and $IC > 12$ and $CNC > 1.26$ and $CFC \leq 3$ Then Moderately difficult to understand

Table 8. Thresholds for the evaluation of modifiability.

Quality metric	Thresholds	Linguistic interpretation
CNC	$CNC \leq 1.30$	Very low
	$1.30 < CNC \leq 1.54$	Low
	$1.54 < CNC \leq 1.61$	Moderate
	$1.61 < CNC \leq 1.85$	High
	$CNC > 1.85$	Very high
NOAJS	$NOAJS \leq 7$	Very low
	$7 < NOAJS \leq 17$	Low
	$17 < NOAJS \leq 33$	Moderate
	$33 < NOAJS \leq 55$	High
	$NOAJS > 55$	Very high
CFC	$CFC \leq 3$	Low
	$3 < CFC \leq 9$	Moderate
	$CFC > 9$	High
NOA	$NOA \leq 6$	Very low
	$6 < NOA \leq 12$	Low
	$12 < NOA \leq 26$	Moderate
	$26 < NOA \leq 44$	High
	$NOA > 44$	Very high
D	$D \leq 0.043$	Low
	$D > 0.043$	High
CP	$CP \leq 0.032$	Low
	$0.032 < CP \leq 0.077$	Moderate
	$0.077 < CP \leq 0.09$	High
	$CP > 0.09$	Very high
HPC_V	$HPC\_V \leq 14$	Low
	$14 < HPC\_V \leq 53$	Moderate
	$HPC\_V > 53$	High
HPC_D	$HPC\_D \leq 5.25$	Low
	$HPC\_D > 5.25$	High

Table 9. Excerpt of decision rules to assess the level of modifiability [32].

	Decision rule
R1	If $CFC \leq 9$ and $HPC\_V \leq 53$ and $NOA \leq 6$ Then Easy to modify
R2	If $HPC\_V \leq 53$ and $NOA > 6$ and $CFC \leq 3$ and $NOA \leq 12$ and $CP \leq 0.077$ Then Easy to modify
R3	If $NOA \leq 26$ and $CFC > 3$ and $NOA > 12$ and $HPC\_V \leq 14$ Then Moderately difficult to modify

of membership functions that we defined based on the identified approximate thresholds (Sect. 3.5). We defined one membership function for each possible fuzzy set per quality metric (Sect. 3.5).

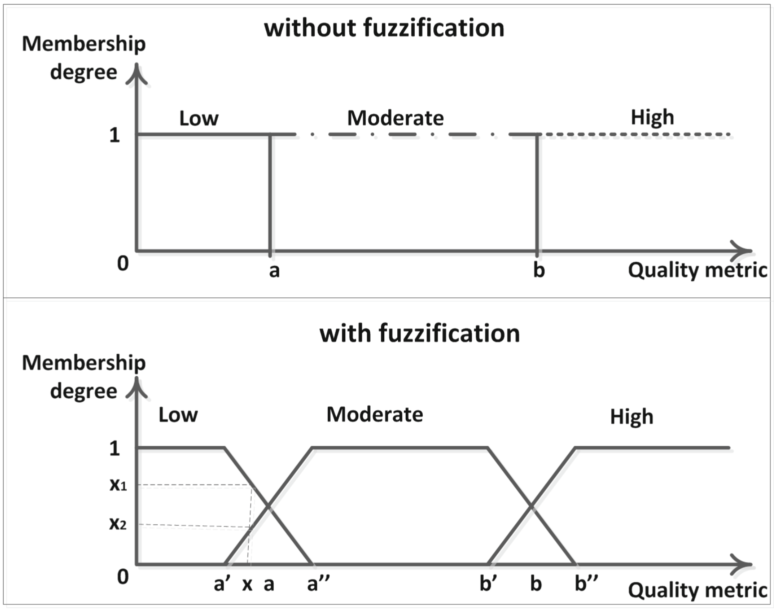


Fig. 1. Membership function definition [32].

In the first part of Fig. 1 (i.e., without fuzzification), *a* and *b* values represent the approximate thresholds determined through the use of decision trees and fuzzy sets associated with the different intervals fixed by experts (i.e., low, moderate, and high). In this figure, each value of a quality metric can belong only to one single fuzzy set with a membership degree equals to 1. This case is true when the fixed thresholds are exact and precise. However, because it is not the case of the thresholds defined in this paper, we use the membership function depicted in the second part of Fig. 1 (i.e., with fuzzification). The values of *a'*, *a"*, *b'*, and *b"* are defined by experts for each quality metric. Each value within [a', a"] and [b', b"] intervals belong to two fuzzy sets with different membership degrees. For example, the value "x" belongs to the two fuzzy sets "low" and "moderate" with membership degree "*x1*" and "*x2*", respectively.

4.2 Inference

The inference is based on a set of fuzzy decision rules written in a natural language and according to a specific syntax "if X is A and/or Y is B then Z is C", where X and Y are input variables, Z is an output variable, and A, B, C are their corresponding linguistic values. We defined fuzzy decision rules to determine the comprehensibility and modifiability levels of a BP model based on the set of quality metrics. We defined the required fuzzy decision rules based on the rules obtained from the decision tree (Sect. 3.5). To this end, we replaced the crisp values with their corresponding linguistic values and rewrote the rules according to the syntax required by fuzzy logic. The total number of defined fuzzy decision rules is 210 rules for assessing comprehensibility and 260 for assessing modifiability. Tables 10 and 11, respectively, depict an extract of the defined fuzzy decision rules for comprehensibility and modifiability.

Table 10. Excerpt of fuzzy decision rules to assess the comprehensibility level [32].

	Fuzzy decision rule
FR1	IF IC IS Low THEN ComprehensibilityLevel IS EasyToUnderstand
FR2	IF IC IS Moderate AND CNC IS Low THEN ComprehensibilityLevel IS EasyToUnderstand
FR3	IF IC IS Moderate AND CNC IS Moderate AND CFC IS Low THEN ComprehensibilityLevel IS ModeratelyDifficultToUnderstand

Table 11. Excerpt of fuzzy decision rules to assess the modifiability level [32].

	Fuzzy decision rule
FR1	IF CFC IS Moderate AND HPC_V IS Moderate AND NOA IS VeryLow THEN ModifiabilityLevel IS EasyToModify
FR2	IF HPC_V IS Moderate AND NOA IS VeryLow AND CFC IS Low AND CP IS Moderate THEN ModifiabilityLevel IS EasyToModify
FR3	IF HPC_V IS High AND CFC IS Low AND NOA IS Low AND CP IS Moderate THEN ModifiabilityLevel IS ModeratelyDifficultToModify

4.3 Defuzzification

Defuzzification is the process which converts the fuzzy value of the output variable (i.e., comprehensibility or modifiability level), obtained by the inference engine, into a crisp value. To do so, it aggregates the fuzzy outputs of all the activated fuzzy decision rules to a one fuzzy set, which will be transformed into a crisp value.

In the literature, there are several defuzzification techniques like: center of gravity (used in our work), center average, and maximum. For the center of gravity technique, the crisp value of the output variable is calculated using the following formula:

$$y* = \frac{\int_U y * \mu(y)\,\mathrm{d}y}{\int_U \mu(y)\,\mathrm{d}y} \tag{10}$$

where μ is the universe of discourse that considers all the output values according to the activated fuzzy decision rules.

Defuzzification determines the level of comprehensibility or modifiability of a BP model as well as the degree of certainty of this level. For example, a BP model can be *easy to understand with a certainty degree of 70%.*

5 System Development: BP-FuzzQual

This section introduces the proposed system as well as its experimental evaluation.

5.1 Architecture

We developed a system called BP-FuzzQual that stands for assessing the quality of BP models. It is developed in Java with Jdom and JFuzzyLogic libraries and under eclipse framework. BP-FuzzQual's functional architecture is shown in Fig. 2. A complete video demonstrating the different steps of the BP quality assessment using our system is available at: https://youtube/qaCDjd–_54.

BP-FuzzQual's modules are described below:

– **Parser:** takes as input a BP model in BPMN and determines the crisp values of each used quality metric for estimating either comprehensibility or modifiability of a BP model.
– **Fuzzy Control:** is implemented in Fuzzy Control Language (FCL) which follows the IEC1131 standard, the first international standard for process control software. FCL includes four components: function block interface, fuzzification, rule block, and defuzzification. FCL also allows defining a fifth optional component called optional parameters. The use of the four required components is detailed in what follows:
  • **Function Block Interface:** defines the set of input and output parameters as well as local variables, if required.

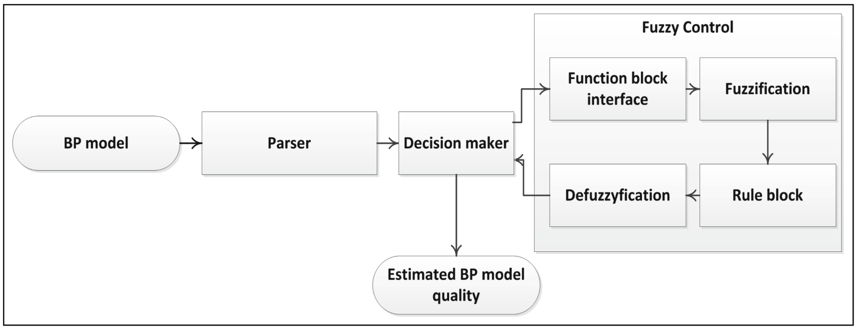


Fig. 2. Functional architecture of BP-FuzzQual [32].

- **Fuzzification:** defines a set of membership functions for each quality metric (Sect. 4.1). Based on these functions, fuzzification converts the crisp values of the quality metric into linguistic values that will be used by the inference engine.
- **Rule Block:** includes the set of fuzzy decision rules that estimate the quality of the BP model (Sect. 4.2).
- **Defuzzification:** converts the linguistic value of the output variable "comprehensibility and modifiability levels" into crisp values. As per Sect. 4.3 the proposed system uses the center of gravity technique to perform this conversion.

– **Decision Maker:** runs the FCL code in order to estimate the quality of the BP model. This module takes as input the crisp values produced by the parser module and communicates them to the fuzzy control module. The decision maker module is developed using java language.

5.2 Experiments

We carried out two sets of experiments. The first set involved anonymous students from our faculty and the second was done using BP-FuzzQual. These experiments were supported by additional resources listed below:

– **Business Process Model:** we used the BP model depicted in Fig. 3. In this model, we use abstract labels in tasks and pools in order to bypass the complexity that could be caused by the business domain.
– **Participants:** the experiments are done in conjunction with 60 person who are mainly students from our faculty. Indeed, we involved Ph.D. students in computer science, students in research master, professional master and fundamental license in computer science during these experiments.
– **Comprehensibility Exercise:** participants had to respond to a set of multiple-choice questions to assess if they understand the BP model. At the end of the exercise, each participant must choose a level of comprehensibility for this model from the three possible levels: easy to understand, moderately

difficult to understand or difficult to understand. The exercise is available at: https://sites.google.com/site/bposcteam2015/ressources.

– **Modifiability Exercise:** participants had to make changes in the BP model depicted in Fig. 3. Indeed, they had to delete or add activities, events, and gateways. At the end of the exercise, each participant had to choose a level of difficulty to modify the model among the three possible levels: easy to modify, moderately difficult to modify or difficult to modify. The exercise is available at: https://sites.google.com/site/bposcteam2015/ressources.

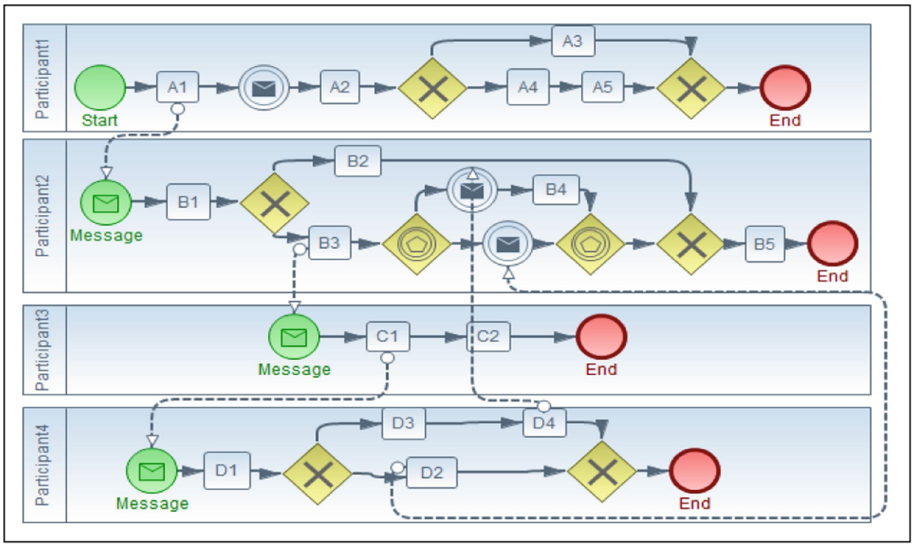


Fig. 3. Example of BP model in BPMN [32].

Experiment 1. Figure 4 represents the number of correct and incorrect answers for the first exercise. In this figure, 78% of the responses are correct showing that the majority of students understood the BP model. This is also confirmed through their responses to the last question of the first exercise, which is about their ability to understand the BP model. Indeed, as depicted in Fig. 5, 76% of students considered the BP model as moderately difficult to understand, 14% as easy to understand, and 10% as difficult to understand.

Figure 6 represents the number of correct and incorrect answers for the second exercise. In this figure, 69% of the responses are correct showing that the majority of students have correctly modified the BP model. In addition, as depicted in Fig. 7 53% of students considered the BP model as moderately difficult to modify, 33% as difficult to modify, and 14% as easy to modify.

Experiment 2. Uses BPMN2 modeler to design the BP model as well as our tool to estimate the comprehensibility and modifiability levels of the BP model presented in Fig. 3.

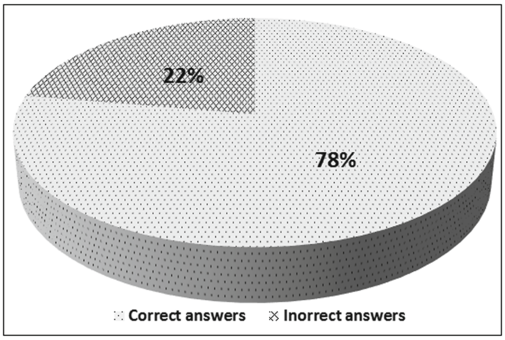


Fig. 4. Correct and incorrect answers for comprehensibility assessment [32].

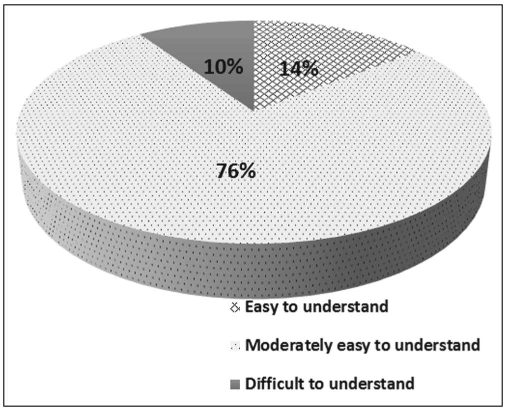


Fig. 5. Students' judgments about the BP model comprehensibility-level [32].

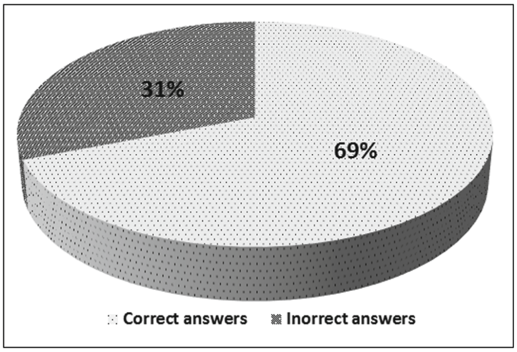


Fig. 6. Correct and incorrect answers for modifiability assessment [32].

For instance, the estimated comprehensibility level of the BP model of Fig. 3 is "*Moderately difficult to understand with a certainty degree of 63%*". Figure 8 shows the interface for comprehensibility assessment.

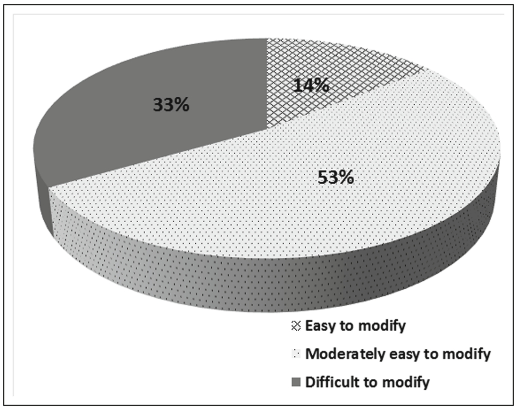
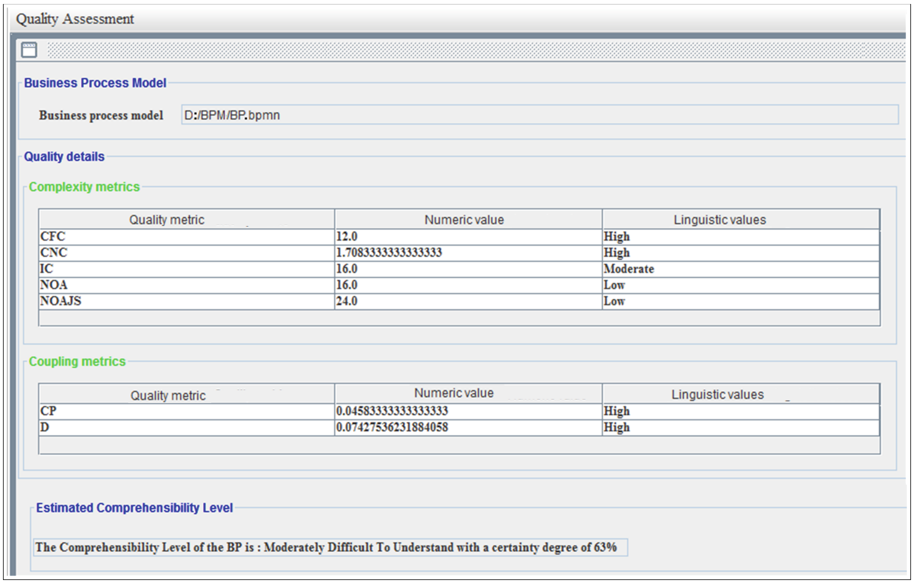


Fig. 7. Students judgment about the BP model modifiability-level [32].


Quality Assessment

Business Process Model

Business process model D:/BPM/BP.bpmn

Quality details

Complexity metrics

Quality metric	Numeric value	Linguistic values
CFC	12.0	High
CNC	1.7083333333333333	High
IC	16.0	Moderate
NOA	16.0	Low
NOAJS	24.0	Low

Coupling metrics

Quality metric	Numeric value	Linguistic values
CP	0.04583333333333333	High
D	0.07427536231884058	High

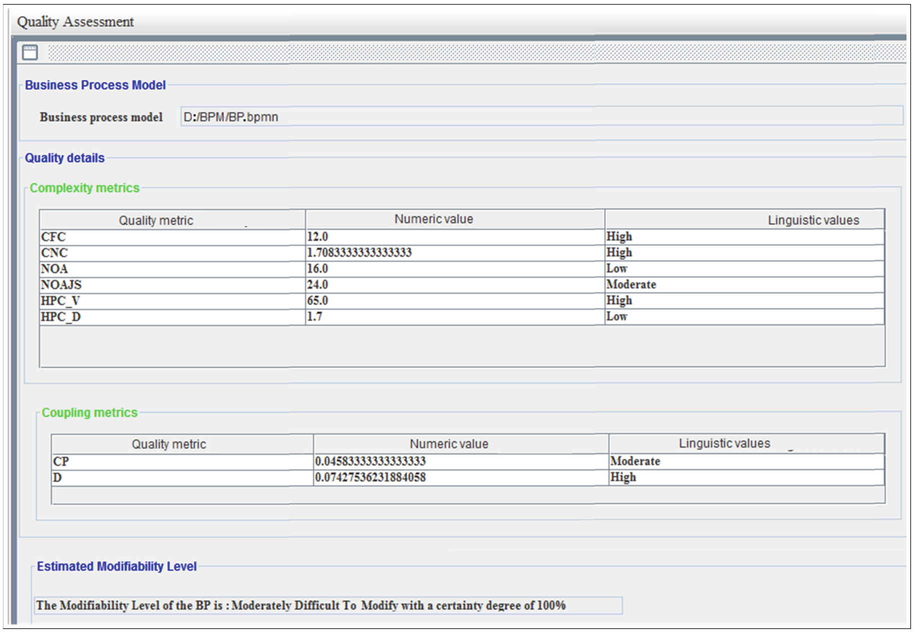
Estimated Comprehensibility Level

The Comprehensibility Level of the BP is : Moderately Difficult To Understand with a certainty degree of 63%

Fig. 8. Comprehensibility assessment interface [32].

For instance, the estimated modifiability level of the BP model of Fig. 3 is "*Moderately difficult to modify with a certainty degree of 100%*". Figure 9 shows the interface dedicated for modifiability assessment.

To sum up, the experiments show that the students considered the BP model as moderately difficult to understand; this is proved by their responses to the comprehensibility questions. This is in line with the evaluation done by BP-FuzzQual, which considers the BP model as moderately difficult to understand. Similarly, when dealing with modifiability, the students consider the BP model as



Quality Assessment

Business Process Model

Business process model: D:/BPM/BP.bpmn

Quality details

Complexity metrics

Quality metric	Numeric value	Linguistic values
CFC	12.0	High
CNC	1.7083333333333333	High
NOA	16.0	Low
NOAJS	24.0	Moderate
HPC_V	65.0	High
HPC_D	1.7	Low

Coupling metrics

Quality metric	Numeric value	Linguistic values
CP	0.04583333333333333	Moderate
D	0.07427536231884058	High

Estimated Modifiability Level

The Modifiability Level of the BP is : Moderately Difficult To Modify with a certainty degree of 100%

Fig. 9. Modifiability assessment interface [32].

around moderately difficult to modify and difficult to modify while BP-FuzzQual considers that the BP model is moderately difficult to modify. Overall, these conforming results show that our approach produces encouraging results that should be proved through additional experiments.

6 Conclusion

The quality of a Business Process (BP) model is critical to the successful achievement of this model's lifecycle phases. The higher the quality is, the "easier" the execution and evaluation of these phases become. The literature refers to different techniques that aim at improving a BP model quality [13,14,17]. Some techniques assist BP engineers develop high-quality models while others propose metrics to assess the quality. However, despite all these initiatives, there is not a common framework for assessing BP model quality. This lack of a common framework is due, among other things, to the absence of a consensus on what a "good" model is.

In this paper, we proposed a fuzzy-based approach for assessing the quality of BPMN-based BP models with emphasis on two quality characteristics that are comprehensibility and modifiability. The approach is based on the ISO/IEC 25010 standard [9] along with a set of quality metrics like CFC, NOA, and NOAJS reported in the BP model quality literature. In addition, for a concise interpretation of the quality metrics, the approach uses data mining techniques (decision tree) to determine thresholds that high-quality BP models should attain and/or maintain. To tackle the approximate nature of these thresholds during BP model quality assessment, we beefed-up our approach with fuzzy logic. Furthermore, we automated the assessment process with an in-house Java-based tool that calculates the values of the different metrics. Based on these values the tool determines the comprehensibility and modifiability levels of a BP model. The preliminary experiments' results are very supportive of mixing data mining and fuzzy logic for better assessment of BP model quality.

In terms of future work, we would like to examine the integration of other ISO/IEC 25010 quality metrics and characteristics, like compatibility and reliability, into our approach. Developing recommendations to BP engineers for higher quality BP models, is also part of our future work.

References

1. Bender, R.: Quantitative risk assessment in epidemiological studies investigating threshold effects. Biom. J. **41**(3), 305–319 (1999)
2. Cardoso, J., Mendling, J., Neumann, G., Reijers, H.A.: A discourse on complexity of process models. In: Eder, J., Dustdar, S. (eds.) BPM 2006. LNCS, vol. 4103, pp. 117–128. Springer, Heidelberg (2006). https://doi.org/10.1007/11837862_13
3. Cardoso, J., Vanderfeesten, I., Reijers, H.A.: Computing coupling for business process models (2010). http://eden.dei.uc.pt/~jcardoso/Research/Papers/Old%20paper%20format/Caise-19th-Coupling-Cardoso-Vanderfeesten.pdf. Accessed 20 Sept 2012
4. Curtis, B., Kellner, M.I., Over, J.: Process modeling. Commun. ACM **35**(9), 75–90 (1992)
5. Gucegliogl u, A.S., Demirors, O.: Using software quality characteristics to measure business process quality. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 374–379. Springer, Heidelberg (2005). https://doi.org/10.1007/11538394_26
6. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. ACM SIGKDD Explor. Newsl. **11**(1), 10–18 (2009)
7. Hanley, J.A., McNeil, B.J.: The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology **143**(1), 29–36 (1982)
8. Heinrich, R.: Aligning business process quality and information system quality. Ph.D. thesis (2013)
9. ISO: ISO/IEC 25010:2011 - systems and software engineering - systems and software quality requirements and evaluation (square) - system and software quality models (2011). https://www.iso.org/standard/35733.html. Accessed 08 Dec 2016
10. ISO: ISO/IEC 19510:2013 - information technology - object management group business process model and notation (2013). http://www.iso.org/iso/catalogue_detail.htm?csnumber=62652. Accessed 17 Oct 2016
11. Makni, L., Khlif, W., Haddar, N.Z., Ben-Abdallah, H.: A tool for evaluating the quality of business process models. In: ISSS/BPSC, pp. 230–242. Citeseer (2010)
12. Mendling, J.: Testing density as a complexity metric for EPCs. In: German EPC Workshop on Density of Process Models (2006)

13. Mendling, J., Reijers, H.A., van der Aalst, W.M.: Seven process modeling guidelines (7PMG). Inf. Softw. Technol. **52**(2), 127–136 (2010)
14. Mendling, J., Sánchez-González, L., Garcia, F., La Rosa, M.: Thresholds for error probability measures of business process models. J. Syst. Softw. **85**(5), 1188–1197 (2012)
15. Morimoto, S.: A survey of formal verification for business process modeling. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008. LNCS, vol. 5102, pp. 514–522. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69387-1_58
16. Muketha, G., Ghani, A., Selamat, M., Atan, R.: A survey of business process complexity metrics. Inf. Technol. J. **9**(7), 1336–1344 (2010)
17. de Oca, I.M.M., Snoeck, M., Reijers, H.A., Rodríguez-Morffi, A.: A systematic literature review of studies on business process modeling quality. Inf. Softw. Technol. **58**, 187–205 (2015)
18. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
19. Reijers, H.A., Vanderfeesten, I.T.P.: Cohesion and coupling metrics for workflow process design. In: Desel, J., Pernici, B., Weske, M. (eds.) BPM 2004. LNCS, vol. 3080, pp. 290–305. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-25970-1_19
20. Sadowska, M.: An approach to assessing the quality of business process models expressed in BPMN. e-Inf. Softw. Eng. J. **9**(1), 57–77 (2015)
21. Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier methodology. IEEE Trans. Syst. Man Cybern. **21**(3), 660–674 (1991)
22. Sánchez-González, L., García, F., Mendling, J., Ruiz, F.: Quality assessment of business process models based on thresholds. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6426, pp. 78–95. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16934-2_9
23. Sánchez-González, L., García, F., Ruiz, F., Mendling, J.: Quality indicators for business process models from a gateway complexity perspective. Inf. Softw. Technol. **54**(11), 1159–1174 (2012)
24. Sánchez-GonzáLez, L., GarcíA, F., Ruiz, F., Piattini, M.: Toward a quality framework for business process models. Int. J. Coop. Inf. Syst. **22**(01), 1350003 (2013)
25. Sánchez-González, L., Ruiz, F., García, F., Cardoso, J.: Towards thresholds of control flow complexity measures for BPMN models. In: Proceedings of the 2011 ACM symposium on Applied computing, pp. 1445–1450. ACM (2011)
26. Schmidt, R., Nurcan, S.: Augmenting BPM with social software. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 201–206. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12186-9_19
27. Vanderfeesten, I., Cardoso, J., Mendling, J., Reijers, H.A., van der Aalst, W.M.: Quality metrics for business process models. BPM Workflow Handb. **144**, 179–190 (2007)
28. Vanderfeesten, I., Reijers, H.A., Mendling, J., van der Aalst, W.M.P., Cardoso, J.: On a quest for good process models: the cross-connectivity metric. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 480–494. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69534-9_36
29. Vanderfeesten, I.T., Cardoso, J., Reijers, H.A.: A weighted coupling metric for business process models. In: CAiSE Forum, vol. 247 (2007)
30. Watahiki, K., Ishikawa, F., Hiraishi, K.: Formal verification of business processes with temporal and resource constraints. In: 2011 IEEE International Conference on Systems, Man, and Cybernetics, SMC, pp. 1173–1180. IEEE (2011)

31. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28616-2
32. Yahya, F., Boukadi, K., Ben-Abdallah, H., Maamar, Z.: A fuzzy logic-based approach for assessing the quality of business process models. In: Proceedings of the 12th International Conference on Software Technologies - Volume 1, ICSOFT, pp. 61–72. INSTICC, SciTePress (2017)
33. Zadeh, L.A.: Fuzzy sets. Inf. Control **8**(3), 338–353 (1965)
34. Zadeh, L.A.: Is there a need for fuzzy logic? Inf. Sci. **178**(13), 2751–2779 (2008)

Solving Multiobjective Knapsack Problem Using Scalarizing Function Based Local Search

Imen Ben Mansour(✉), Ines Alaya, and Moncef Tagina

National School of Computer Sciences,
University of Manouba, 2010 Manouba, Tunisia
{imen.benmansour,ines.alaya,moncef.tagina}@ensi-uma.tn

Abstract. Multiobjective optimization has grown to become an active research area since almost all real-world problems have multiple, usually conflicting, objectives. In this paper, we focus on the multiobjective knapsack problem, we solve instances with two, three and four objectives from the literature. We define an iterated local search algorithm in which a Tchebycheff function is used as a selection process to generate a good approximation of the efficient set. The proposed algorithm, *Min-Max* TLS, designs an efficient neighborhood function based on a permutation process. *Min-Max* TLS is compared with state-of-the-art approaches such as 2PPLS and MOTGA. Results show that our algorithm can achieve a good balance between exploitation and exploration during the search process.

Keywords: Multiobjective knapsack problem · Iterated local search Scalarization functions · Tchebycheff functions

1 Introduction

Many real-life optimization problems can hardly be formulated as a mono-objective problem, which explains the permanent growing interest in the field of multiobjective optimization. For instance, in transportation problem there is more than one objective to optimize. The cost of the transport, the duration and the capacity of the transport, all of this could be objectives to be optimized. In such problems, usually optimizing one objective, leads to degrading other objectives. Thus, finding a good trade-off between several conflicting objectives is one of the main goals of the multiobjective optimization problems (MOPs). So, it consists in optimizing simultaneously several conflicting objectives in order to find a set of solutions called Pareto front or set or non-dominated set.

The multiobjective multidimensional knapsack problem (MOMKP) which is one of the hardest multiobjective combinatorial optimization problems, presents a formal model for many real-world problems. It can be modelized as resource allocation [1], portfolio optimization [2] and budget allocation [3]. Moreover,

E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 210–228, 2018.
https://doi.org/10.1007/978-3-319-93641-3_10

MOMKP can be modelized as a subproblem such as the flight crew choice [4] and many other general integer programs. The main goal of MOMKP consists in selecting a subset of items in order to maximize m objective functions with respect to q resource constraints. To solve this NP-hard problem, several approaches based on metaheuristics were proposed in the literature: In [5], Zitzler et al. introduced the well-known SPEA2 which is an elitist algorithm, based on a ranking dominance procedure. Deb et al. proposed the NSGAII [6], another well-known multiobjective evolutionary algorithm that is also an elitist approach and uses a different ranking dominance procedure. In [7], a memetic algorithm integrates a tabu search method, called MEMOTS, was devised. A generic ACO algorithm (m-ACO) was presented and instantiated with four variants In [8]. m-ACO is parameterized by the number of ant colonies and the number of the pheromone structures. Recently, an indicator based ant colony approach which is abbreviated as IBACO is proposed in [9]. The algorithm uses the indicator optimization principle to guide ants during their search in the most promising areas by reinforcing the best solutions by rewarding pheromone.

Related Works. Local search approaches have been widely employed on the NP-hard multiobjective optimization problems, including problems formulated as a multiobjective multidimensional knapsack problem [10–12]. Indeed, its simple and practical appearance has enabled it to be efficient in many real-world problem such as railway transportation, airline operations, portfolio optimization and computer networks [13]. Motivated by the success of these methods, several papers propose scalarization-based local search methods for MOMKP. These methods have received a great interest from the scientific community because of their success for solving multiobjective problems. They represent a simple way to transform a multiobjective problem into one single or a family of single objective optimization problems. In [14], Lust and Teghem present the two-phase Pareto local search (2PPLS). The algorithm uses a Pareto-based local search combined with a very-large scale neighborhood method based on vectors weight. The 2PPLS combines Pareto local search and aggregation to solve MOMKP, it consists of two main phases. Phase 1 generates an approximation of all the supported solutions by solving a number of linear aggregation problems. Phase 2 applies a Pareto local search to every solution generated in Phase 1 to find non-supported Pareto optimal solutions. In [15], a genetic algorithm based on Tchebycheff scalarization function called MOTGA has been proposed. The algorithm was applied to the MOMKP problem. The Pareto front is divided into a few small parts. Each part is approximated by a stage and each stage is guided by a different Tchebycheff aggregation function. More recently, a memetic algorithm based on decomposition (MOMAD) was proposed in [16]. MOMAD combines the ideas of the two approaches: the 2PPLS and the evolutionary algorithm MOEA/D introduced in [17]. In MOMAD, a Pareto local search method is first applied for the neighborhood search then a single objective local search is applied to each perturbed solution.

Motivations and Contributions. From the previously mentioned state-of-the-art algorithms, one can say that scalarization and weighted based methods present promising approaches to solve MOMKP. In the literature, various efforts have been directed toward the use of the metaheuristics to solve the MOMKP. Generally, these approaches present a good way to solve NP-hard problems. However, if a heuristic searching method is used these approaches may be very time consuming, especially for the large size instances. So, the challenge is optimizing both the quality results and the time-consuming. In this work, we propose an iterated multiobjective local search based on Tchebycheff function: *Min-Max* TLS for MOMKP. In order to investigate the impact of the choice of the Tchebycheff function on the selection process, the two well-known Tchebycheff functions are studied in this work: the weighted Tchebycheff: WT [18] and the augmented weighted Tchebycheff: AugWT [19]. Moreover, one of the key aspects that has to be considered is the neighborhood structure since it plays a central role in a local search algorithm. *Min-Max* TLS integrates an efficient weighted neighborhood structure called *Min-Max* $\mathcal{N}(s)$ to explore the neighborhood. *Min-Max* $\mathcal{N}(s)$ allows to establish an order between items according to their profitability. It has the goal of finding the item that minimizes an extraction ratio to permute it with items that maximize an insertion ratio in order to improve the quality of the obtained solutions in a small computational time. Furthermore, in weighted based methods, the generation of the weight vectors presents a significant point and influences the quality of the generated solutions. Thus, *Min-Max* TLS defines a weight vector generation method called Gw: **G**radual **w**eight vector generation method. Gw creates a set of weight vectors, corresponding to search directions, guiding the search gradually on almost all regions of the Pareto front.

The paper is organized as follows. In the next section, we define the multiobjective optimization problems. In Sect. 3, we present the multiobjective multidimensional knapsack problem. In Sect. 4, the Tchebycheff functions used in this paper are described. Section 5 presents our main contributions, the proposed algorithm and its different main functions. Then, in Sect. 6, we discuss the experimental results obtained from *Min-Max* TLS. Finally, in Sect. 7, we end the paper with the conclusion and our perspectives.

2 Multiobjective Optimization Problems

Before introducing the MOMKP, let us introduce some useful notations and definitions related to a general multiobjective optimization problem.

Let X denote the decision space of a general optimization problem, Z the corresponding objective space and m objective functions $f_1, f_2, \ldots, f_m$ that assign to each decision vector $x \in X$ one objective vector $z = (f_1(x), f_2(x) \ldots, f_m(x)) \in Z$. In the following, we assume that all objective functions are to be maximized:

Definition 1. *A decision vector $x \in X$ is said to dominate another decision vector $x' \in X$, noted $x \succ x'$, iff $\forall i \in \{1, 2, \ldots, m\}$, $f_i(x) \geq f_i(x')$ and $\exists j \in \{1, 2, \ldots, m\}, f_j(x) > f_j(x')$;*

Definition 2. *A decision vector x is called weakly non-dominated, if there exists no $x' \in X$ such that for all $i \in \{1, 2, \ldots, m\}, f_i(x') > f_i(x)$;*

Definition 3. *A decision vector x is Pareto-optimal or non-dominated, if a decision vector x' which dominates x does not exist.*

The goal of a MOP is to find the set of all non-dominated solutions called Pareto set. When using a metaheuristic approach, the goal is to find a Pareto set approximation.

3 MOMKP Formulation

The multiobjective multidimensional knapsack problem could be formulated as follows:

$$Maximize \sum_{j=1}^{n} p_j^k x_j \qquad k = 1, 2, \ldots, m \tag{1}$$

$$Subject\,to \sum_{j=1}^{n} w_j^i x_j \leq b_i \qquad i = 1, 2, \ldots, q \tag{2}$$

$$x_j \in \{0, 1\} \qquad j = 1, 2, \ldots, n$$

where n is the number of items, for each item I_j is assigned a decision variable x_j equal to 1 if the item is selected, 0 otherwise. Each item I_j has a profit p_j^k relatively to the objective k and a weight w_j^i relatively to the resource i. The aim of the problem is to select a subset of items in order to maximize m objective functions while not exceeding q resource constraints. b_i is the total quantity available for the resource i.

In this paper, the considered instances assume that the number of objective functions is equal to the number of resources ($m = q$).

4 Tchebycheff Functions

For decomposing a multiobjective optimization problem into one single-objective many scalarization functions exist. In following, we introduce two among the most commonly-used methods based on the Tchebycheff metric:

4.1 The Weighted Tchebycheff Function

$$\begin{array}{c} minimize\, W^T(x|\lambda, z^*) = \underset{1 \leq k \leq m}{max} \{\lambda^k |z_k^* - f_k(x)|\} \\ Subject\,to \quad x \in X \end{array} \tag{3}$$

where x is the solution to evaluate, $\lambda = (\lambda^1, \lambda^2, \ldots, \lambda^m)$ is the weight vector, such that $\lambda^k \geq 0$ for all $k = 1, 2, \ldots, m$ and $\sum_{k=1}^{m} \lambda^k = 1$. The ideal point $z^* = (z_1^*, z_2^*, \ldots, z_m^*)$ in the objective space is used as a reference point, calculated as follows:

$$z_k^* = max\{f_k(x)|x \in X\},\ k = 1, 2, \ldots, m \tag{4}$$

4.2 The Augmented Weighted Tchebycheff Function

One of the main advantage of the weighted Tchebycheff method is that by varying appropriately the weight vectors and/or the reference point, every non-dominated solution of the Pareto front of the studied MOP, explicitly includes non-convex and discrete problems, can be found [20]. On the other hand, the weighted Tchebycheff method generates also the weakly non-dominated solutions which is often undesirable in MOP. In [19], Steuer and Choo suggested to add an l_1-term, parameterized by ε, to $WT(x|\lambda, z^*)$ that helps to avoid the weakly non-dominated solutions. The resulting function is defined as follows:

$$\begin{aligned} Aug^{WT}(x|\lambda, z^*) = minimize\, W^T(x|\lambda, z^*) + \varepsilon \sum_{k=1}^{m} \{\lambda^k |z_k^* - f_k(x)|\} \\ Subject\, to \quad x \in X \end{aligned} \tag{5}$$

where $\varepsilon \geq 0$ is usually chosen as a small positive constant and z^* is calculated as defined in Eq. 4.

5 Our Proposed Approach *Min-Max* TLS: *Min-Max* Tchebycheff Based Local Search

The proposed approach *Min-Max* TLS is detailed in the following:

```
Begin
 $\mathcal{V} \leftarrow Generate\ Weight\ Vectors(H)$
 Repeat
     $\mathcal{P} \leftarrow Perturbation(\mathcal{P},\ N)$
     $\mathcal{A} \leftarrow Non\text{–}dominated\ solutions(\mathcal{P})$
     $\lambda \leftarrow Select\ Weight\ Vector(\mathcal{V},\ H)$
     ForAll $(s \in \mathcal{P})$ do
        Repeat
           $UpdateReferencePoint$
           $s^* \leftarrow Neighborhood(s)$
           If $(Acceptance(s^*,\ w,\ \mathcal{P}))$ then
              $\mathcal{P}' \leftarrow Replace(s^*,\ w,\ \mathcal{P})$
           EndIf
        Until $s^* \neq w$
     EndForAll
     $\mathcal{A} \leftarrow Non\text{–}dominated\ solutions(\mathcal{A} \cup \mathcal{P}')$
 Until($Tmax$ is reached)
End
```

Let $\mathcal{P}$ be the current population of *Min-Max* TLS of size N. Initially, a weight vector is selected from the set $\mathcal{V}$ of H weight vectors, generated according to the proposed Gw method. Then, for each solution s of $\mathcal{P}$ the neighborhood is explored following the proposed weighted neighborhood structure *Min-Max*

$\mathcal{N}(s)$, until a good solution is found i.e. one which is better than at least the worst solution w in $\mathcal{P}$ in terms of the used Tchebycheff function. This process is iterated until all solutions in $\mathcal{P}$ are explored. All the non-dominated solutions found during the local search step are stored in the archive $\mathcal{A}$. The algorithm stops when a maximum number of iterations T_{max} is met and the archive $\mathcal{A}$ is returned.

5.1 Weight Vectors Generation

In order to generate the set $\mathcal{V}$, we propose the Gw method: Let $\lambda = (\lambda^1, \lambda^2, \ldots, \lambda^m)$ be the weight vector, such that $\lambda^k \geq 0$ for all $k = 1, 2, \ldots, m$ and $\sum_{k=1}^{m} \lambda^k = 1$. For the bi-objective case, the weight vectors are calculated as follow:

$$\lambda^1(t) = \ln\left[\left(\frac{4 * t * e}{FQ}\right) + \cos\left(\frac{2 * \pi * t}{FQ}\right)\right] \tag{6}$$

$$\lambda^2(t) = 1.0 - \lambda^1(t) \tag{7}$$

where t is the iteration index and e is exp (1). The weight vectors are controlled by FQ, which is the weight change frequency. If the FQ is well set, the population can move smoothly from one point to another thus whole regions of the objective space can be explored and different Pareto-optimal points covering the Pareto front can be found through this appropriate variation of research orientations. Moreover, FQ is used to set the number of weight vectors to be generated. Let $FQ = 800$ and $m = 2$, $H = (FQ/4)^{m-1} = 200$. Figure 1 shows an example of how the weight vector values λ^1 and λ^2 change within 200 iterations and FQ set to 800 during the local search process.

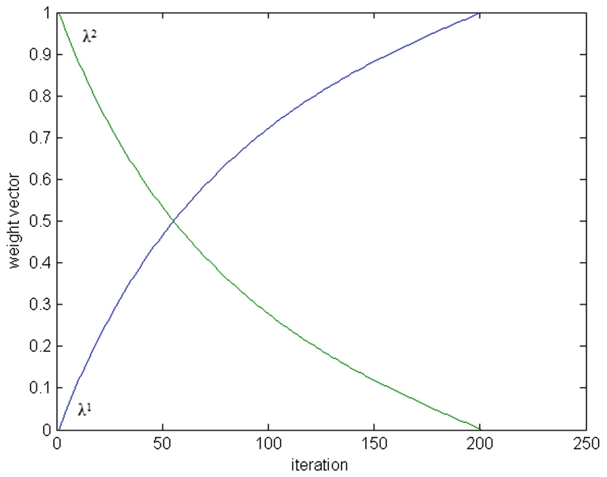


Fig. 1. The weight vector value change within $T_{max} = 200$ and $FQ = 800$ [21].

The extension of the Gw method with more than two objectives is theoretically straightforward. In the following, we describe the method Gw, in case of four objectives ($m = 4$).

```
Begin
 For($i$ from 0 to $FQ/4$) do
     $\lambda^1 = \ln\left[\left(\frac{4*i*e}{FQ}\right) + \cos\left(\frac{2*\pi*i}{FQ}\right)\right]$
     For($j$ from 0 to $FQ/4$) do
       $\lambda^2 = (1.0 - \lambda^1)*\ln\left[\left(\frac{4*j*e}{FQ}\right) + \cos\left(\frac{2*\pi*j}{FQ}\right)\right]$
       For($k$ from 0 to $FQ/4$) do
         $\lambda^3 = (1.0 - \lambda^1 - \lambda^2)*\ln\left[\left(\frac{4*k*e}{FQ}\right) + \cos\left(\frac{2*\pi*k}{FQ}\right)\right]$
         $\lambda^4 = 1.0 - \lambda^1 - \lambda^2 - \lambda^3$
       EndFor
     EndFor
 EndFor
End
```

The algorithm shown above is executed only once before the beginning of the main loop of *Min-Max* TLS. Once the initial population is generated, a weight vector is selected from the set $\mathcal{V}$, according to the iteration index and assigned to the solution to be evaluated in such way that all the member of the population have the same weight vector during the same iteration.

5.2 Initial Population Initialization

Like in most evolutionary approaches, the first initial population in *Min-Max* TLS is randomly created. In *Min-Max* TLS, a solution s is composed of two subsets $I_l^+ = \{I_1^+, I_2^+, \ldots, I_T^+\}$ and $I_l^- = \{I_1^-, I_2^-, \ldots, I_{NT}^-\}$ where I_l^+ is the subset of the items selected in the solution s, T is the number of taken items, I_l^- is the subset of the remaining items (unselected items) and NT is the number of untaken items. To create a first random population, *Min-Max* TLS creates N random solutions. Every solution s is created by selecting a random item I_j among the n items. If any resource capacity is violated by the item I_j, I_j is added to the subset I_l^- otherwise I_j is added to the solution s and to the subset I_l^+. This process is iterated until all items are placed.

5.3 Perturbation

After the update of the archive $\mathcal{A}$, a new population is generated. In order to reach different and new region of the search space, where the exploration starts in the next local search step, we propose to use a perturbation function to generate the new population: At each iteration, N new solutions are randomly selected from the archive $\mathcal{A}$ if the size of $\mathcal{A}$ exceeds N, else if the archive size is less than the size of the population, the missing solutions are created randomly as done in the first initial population. Let μ be the rate of noise, for every solution s we remove randomly $(\mu * T)$ items from the subset I_l^+ and we add them to the subset $I_{\overline{l}}^-$ of the solution s. Then, as long as we do not violate any resource constraints. Items are randomly added from $I_{\overline{l}}^-$ to I_l^+. In fact, this random move allows us to reach new local optima. They can be weak or strong noise which means moving to a near local optima or jump totally to a new region where it is considerably difficult to find a new local optima.

5.4 Update Reference Point

When a new solution is introduced in the population, the reference point z^* is updated. After the initialization process of the population and after each neighborhood exploration, the new maximal value of all objective functions is calculated according to Eq. 4.

5.5 Neighborhood Structure

The generation of the neighborhood is one of the most important part of a local search algorithm. In this paper, we use a proposed weighted neighborhood structure *Min-Max* $\mathcal{N}(s)$. By means of the two proposed algorithms: *Min-Extraction-Item* algorithm and *Max-Insertion-Item* algorithm, *Min-Max* $\mathcal{N}(s)$ tries to gives a better neighbor of a solution s.

Min-Extraction-Item. The *Min-Extraction-Item* algorithm can be formulated as follows:

```
Begin
 For ($l^+$ from 1 to $T$) do
      Compute $U(l^+)$
      Add Item $I_l^+$ to $L_U$
 EndFor
 Sort $L_U$ in ascending order
End
```

For each taken item I_l^+ the ratio $U(l^+)$ is calculated as follows:

$$U(l^+) = \frac{\sum_{k=1}^{m} \lambda^k(t) p_{l^+}^k}{\sum_{i=1}^{q} w_{l^+}^i} \tag{8}$$

where $\lambda^k(t)$ is the weight vector selected from the set $\mathcal{V}$, $\sum_{k=1}^{m} \lambda^k(t) p_{l^+}^k$ is the weighted profit of the item I_l^+ and $\sum_{i=1}^{q} w_{l^+}^i$ is its overall weight. The ratio $U(l^+)$ measures the utility value of each item, the lower this ratio is, the worst the item is. Once the ratio $U(l^+)$ is calculated for all items $I_l^+ = \{I_1^+, I_2^+, \ldots, I_T^+\}$, a list L_U containing all the items I_l^+ is created and sorted in ascending order according to the ratio $U(l^+)$ of each item. So that the worst items that minimize the ratio $U(l^+)$ are placed on the top of the list. The *Min-Extraction-Item* algorithm is executed only when a new solution s is selected for neighborhood exploration.

Max-Insertion-Item. The *Max-Insertion-Item* algorithm is outlined as follows:

```
Begin
 For($\bar{l}$ from 1 to $NT$) do
     Compute $\overline{U}(\bar{l})$
     Add Item $I^-_{\bar{l}}$ to $L_{\overline{U}}$
 EndFor
Sort $L_{\overline{U}}$ in decreasing order
End
```

For each untaken item $I^-_{\bar{l}}$ the ratio $\overline{U}(\bar{l})$, inspired from [22], is calculated. We define $R_i(s) = b_i - \sum_{l=1}^{T} w_l^i$ as the remaining quantity of the resource i after the construction of the solution s. The ratio $\overline{U}(\bar{l})$ of an item $I^-_{\bar{l}}$ can be calculated as follows:

$$\overline{U}(\bar{l}) = \frac{\sum_{k=1}^{m} \lambda^k(t) p_{\bar{l}}^k}{\sum_{i=1}^{q} \left(\frac{w_{\bar{l}}^i}{R_i(s)} \right)} \tag{9}$$

where $p_{\bar{l}}^k$ and $w_{\bar{l}}^i$ are respectively the profit and the weight of the candidate item. The ratio $\overline{U}(\bar{l})$ measures the quality of the candidate item according to the solution s where the higher this ratio is, the better the item is. As done with the taken items in the *Min-Extraction-Item* algorithm, a list $L_{\overline{U}}$ containing all the candidate items $I^-_{\bar{l}}$ is created and sorted in decreasing order according to the ratio $\overline{U}(\bar{l})$ of each item. So that the most profitable items that maximize the ratio $\overline{U}(\bar{l})$ are placed on the top of the list.

Min-Max $\mathcal{N}(s)$. A neighbor s^* is created by removing an item from the list L_U and adding items from the list $L_{\overline{U}}$ as long as no resource constraint is violated. In the following, we describe the neighborhood generation procedure.

```
Begin
 Min–Extraction–Item($T$)
 While($s^*$ is the worst)
      Max–Insertion–Item($NT$)
      $s^* \leftarrow$ Remove $L_U(l^+)$ from $s$
      While(no constraint is violated)
         $s^* \leftarrow$ Add $L_{\overline{U}}(\bar{l})$ to solution $s$
      EndWhile
 EndWhile
End
```

5.6 Acceptance Criterion and Replacement Function

In the local search algorithms, the neighborhood exploration stops at the first better solution or when the entire neighborhood is explored. In this work, we choose to stop the exploration once the first better neighbor is found relatively to the used Tchebycheff function. This choice is guided by one main reason is that this mechanism allows to speed up the convergence of the population.

The neighbor s^* is accepted in the population if its fitness value, calculated according to the Weighted Tchebycheff function (Eq. 3) or the Augmented Weighted Tchebycheff function (Eq. 5), is better than the worst solution w in the current population. If so, s^* replaces the worst solution w. Otherwise the neighbor is rejected, the next item on the list L_U is selected, i.e. the item to be removed from the solution s, and the algorithm outlined above is executed again. This process is iterated until a neighbor s^* better than at least the worst solution w is found.

6 Experimental Results

The experiments carried out consider two versions of *Min-Max* TLS: *Min-Max* TLS$_{WT}$ and *Min-Max* TLS$_{AugWT}$ where the solutions are evaluated using respectively the Weighted Tchebycheff function and the Augmented Weighted Tchebycheff function. In order to evaluate the quality of the proposed algorithm and to have a fair comparison, we choose to compare the two proposed versions of *Min-Max* TLS with scalarizition-local search-based methods: MOMAD [16], 2PPLS [14] and MOTGA [15]. For both of the two versions as well as the compared algorithms, we considered the benchmark instances defined in [23], that are widely used in testing several multiobjective heuristics, containing nine instances with 250, 500 and 750 items, in combination with 2, 3 and 4 objectives.

6.1 Experimental Setup

The used parameter settings for *Min-Max* TLS are adjusted experimentally and presented as follow: The population size N is set to 10, we choose a small rate noise $\mu = 0.05$ to apply on selected solutions in order to focus the search around the most useful solutions. The parameter ε used in the AugWT function is set to 10^{-3} as suggested in [20], to ensure that weakly non-dominated solutions are avoided and that all non-dominated solutions of the studied (potentially non-convex) problem can be found. The change frequency FQ and the maximum number of iteration T_{max} are set according to the Table 1. In order to let the weight vectors change smoothly from one iteration to another, so the whole search space will be explored, the number of generated weight vectors H and the stopping criterion T_{max} are set to $(FQ/4)^{m-1}$. MOMAD, 2PPLS and MOTGA were configured as recommended by their authors, for each algorithm and each instance 30 runs are considered.

6.2 Performance Metrics

In order to evaluate the results generated by the different algorithms, we used three measures:

Table 1. The change frequency FQ and T_{max} setting.

Number of objectives	FQ	T_{max}
2	800	200
3	40	100
4	20	125

- The hypervolume Difference: which computes the difference between a reference set, which corresponds to the set of the non-dominated solutions obtained from the union of all solutions of the different runs, and the set of solutions to be evaluated in terms of hypervolume as defined in [23]. The obtained values have to be as close as possible to zero to prove the efficiency of an algorithm against the other approaches;
- The Mann-Whitney Statistical Test: [24] is applied to the results obtained by the hypervolume difference in order to prove that the difference between two algorithms are statistically significant. We reject the null-hypothesis "no significant difference between the two algorithms A and B" if the P-value is lower than the significance level 5%;
- The summary Attainment Surface: [25] is used to compare the approach behaviors in a graphical way. The attainment surface is defined by the objective vectors that have been attained by at least one approximation set of the tested algorithm. In the following we compare the median (50%) attainment surface of the fronts non-dominated solutions found for 30 runs of all the compared algorithms.

All the computational results were obtained by the performance assessment software package PISA provided in http://www.tik.ee.ethz.ch/sop/pisa/.

6.3 Comparison Results

We performed 30 independent runs of each compared algorithm on all the tested instances. In order to have a good idea of the overall performance of each approach, the average hypervolume is computed for each algorithm and each instance. The hypervolume difference results of the proposed algorithms *Min-Max* TLS$_{WT}$ and *Min-Max* TLS$_{AugWT}$, and the compared algorithms MOMAD, 2PPLS and MOTGA appear in Table 2. $m \times n$ denotes the tested instance, m is the number of objectives, n is the number of items and (N/A) denotes that the results are not available. By analyzing the Table 2, it is clear that our proposed approach performs better than the other compared algorithms. In fact, *Min-Min* TLS gives better hypervolume difference results than MOMAD on all the tested instances and better than MOTGA in eight out of nine instances. Moreover, it seems that the average results of MOMAD and MOTGA decrease according to the size of the problem. For the largest and hardest instances, the quality of the obtained results of MOMAD and MOTGA decease clearly which is not the case of *Min-Max* TLS. Thus, the difference between the compared approaches became much more important with the largest instances. By comparing *Min-Max* TLS with 2PPLS on the three bi-objective instances, the two approaches obtained about the same hypervolume values except on the 2×500 instance where 2PPLS preforms better than *Min-Max* TLS.

Table 2. Hypervolume difference average values of *Min-Max* TLS$_{WT}$, *Min-Max* TLS$_{AugWT}$, MOMAD, 2PPLS and MOTGA.

Instances	*Min-Max* TLS$_{WT}$	*Min-Max* TLS$_{AugWT}$	MOMAD	2PPLS	MOTGA
2×250	2.35E−01	2.52E−01	3.16E−01	2.32E−01	2.72E−01
2×500	1.93E−01	2.16E−01	3.56E−01	1.20E−01	1.41E−01
2×750	2.14E−01	2.13E−01	3.00E−01	2.10E−01	2.90E−01
3×250	2.12E−01	2.21E−01	3.93E−01	N/A	3.29E−01
3×500	2.27E−01	2.16E−01	4.56E−01	N/A	3.96E−01
3×750	1.91E−01	1.96E−01	4.31E−01	N/A	2.90E−01
4×250	2.20E−01	2.18E−01	3.76E−01	N/A	3.79E01
4×500	1.93E−01	1.84E−01	3.97E−01	N/A	3.97E−01
4×750	1.88E−01	1.76E−01	4.59E−01	N/A	4.27E−01

Table 3 represents the results obtained by the Mann-Whitney statistical test. The table contains the number of the algorithm which is statistically outperformed by the corresponding algorithm. For example, on the 2×250 instance, *Min-Max* TLS$_{WT}$ statistically outperformed the algorithm number 3 and the algorithm number 5 which correspond to the MOMAD algorithm and the MOTGA algorithm respectively. "-" means that the corresponding algorithm does not statistically outperforms any other algorithm, as is the case with 2PPLS on the 2×750 instance. By analyzing the results of the Mann-Whitney statistical test shown in the Table 3, the first important information that we can extract from the table is that *Min-Max* TLS outperforms significantly MOMAD and MOTGA on all the largest and hardest instances with 3 and 4 objectives. However, MOMAD and MOTGA never significantly outperform the results returned by the proposed approach on these instances. From the three bi-objective instances, we can observe that *Min-Max* TLS outperforms significantly MOMAD and MOTGA on 2×750 and 2×250 instances. On the 2×500 instance, *Min-Max* TLS outperforms only MOMAD.

Figures 2, 3 and 4 compare the average hypervolume difference of *Min-Max* TLS$_{WT}$ and *Min-Max* TLS$_{AugWT}$ according to the number of objectives for instances with 250, 500 and 750 items respectively. These figures show clearly the behaviors of the two versions. As it was previously remarked, the hypervolume values of *Min-Max* TLS$_{WT}$ are slightly better than *Min-Max* TLS$_{AugWT}$ on the smallest instances (2 and 3 objectives with 250 and 2 objectives with 500 items see Figs. 2 and 3) while with the largest instances with 4 objectives and

Table 3. Mann-Whitney statistical test results of *Min-Max* TLS$_{WT}$, *Min-Max* TLS$_{AugWT}$, MOMAD, 2PPLS and MOTGA.

Instances	*Min-Max* TLS$_{WT}$	*Min-Max* TLS$_{AugWT}$	MOMAD	2PPLS	MOTGA
Algorithm Number	1	2	3	4	5
2×250	3, 5	3	_	_	_
2×500	3	3	_	1, 2	1, 2
2×750	3, 5	3, 5	_	_	_
3×250	3, 5	3, 5	_	N/A	_
3×500	3, 5	3, 5	_	N/A	_
3×750	3, 5	3, 5	_	N/A	_
4×250	3, 5	3, 5	_	N/A	_
4×500	3, 5	3, 5	_	N/A	_
4×750	3, 5	3, 5	_	N/A	_

with 500 and 750 items (Figs. 3 and 4 respectively), *Min-Max* TLS$_{AugWT}$ finds slightly better values than *Min-Max* TLS$_{WT}$. As a conclusion, one can say that, according to Tables 2 and 3 and Figs. 2, 3 and 4, *Min-Max* TLS$_{AugWT}$ slightly outperforms *Min-Max* TLS$_{WT}$ for 5 out of the 9 instances in terms of the average hypervolume difference but statistically there is no significant difference between the two versions: *Min-Max* TLS$_{WT}$ and *Min-Max* TLS$_{AugWT}$.

Table 4 gives the average computational time consumed by each algorithm in seconds. From the table, it is clear that our proposed approach *Min-Max* TLS consumes significant shorter CPU time than the other approaches. In fact, the time consumed by *Min-Max* TLS is shorter than the other approaches for some instances about 8 times. When comparing the CPU time of *Min-Max* TLS$_{WT}$ against *Min-Max* TLS$_{AugWT}$, one can observe that the two algorithms consume almost the same running time. Nevertheless, *Min-Max* TLS$_{AugWT}$ consumes slightly more running time than *Min-Max* TLS$_{WT}$ on the same instances.

Figures 5, 6 and 7 show the median attainment surfaces of the approximation sets returned by the two versions *Min-Max* TLS$_{WT}$ and *Min-Max* TLS$_{AugWT}$, MOMAD, 2PPLS and MOTGA for respectively the bi-objectives instances 2×250, 2×500 and 2×750. The first important remarque that can be observed from these figures is that all the tested approaches provide a well-distributed Pareto front where the obtained points cover almost all the Pareto front. In Fig. 5, the surfaces are almost confused, it is difficult to distinguish the algorithms. For the two other instances, Figs. 6 and 7, the surfaces of MOMAD, 2PPLS and MOTGA are slightly above the surfaces returned by *Min-Max* TLS$_{WT}$ and *Min-Max* TLS$_{AugWT}$ on a small middle region of the Pareto front. While throughout the extremity, both of the surfaces of *Min-Max* TLS$_{WT}$ and *Min-Max* TLS$_{AugWT}$ are close to those of MOMAD, 2PPLS and MOTGA. We note that there is no clear difference between the surfaces obtained by *Min-Max*

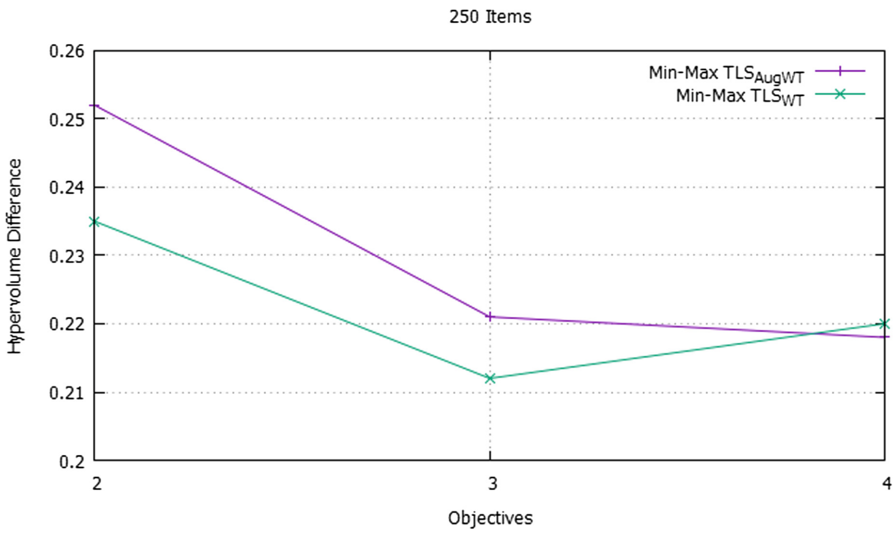


Fig. 2. Average hypervolume difference of *Min-Max* TLS$_{WT}$ and *Min-Max* TLS$_{AugWT}$ with 250 items [21].

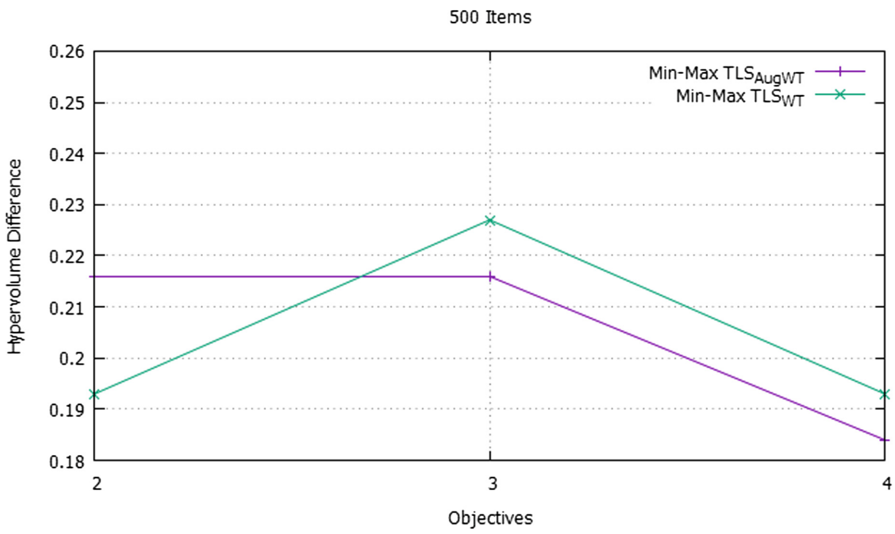


Fig. 3. Average hypervolume difference of *Min-Max* TLS$_{WT}$ and *Min-Max* TLS$_{AugWT}$ with 500 items [21].

TLS$_{WT}$ and *Min-Max* TLS$_{AugWT}$. Thus, these graphics confirm the numerical results obtained previously, where we have found that, generally, for the bi-objective instances the compared algorithms are not significantly different.

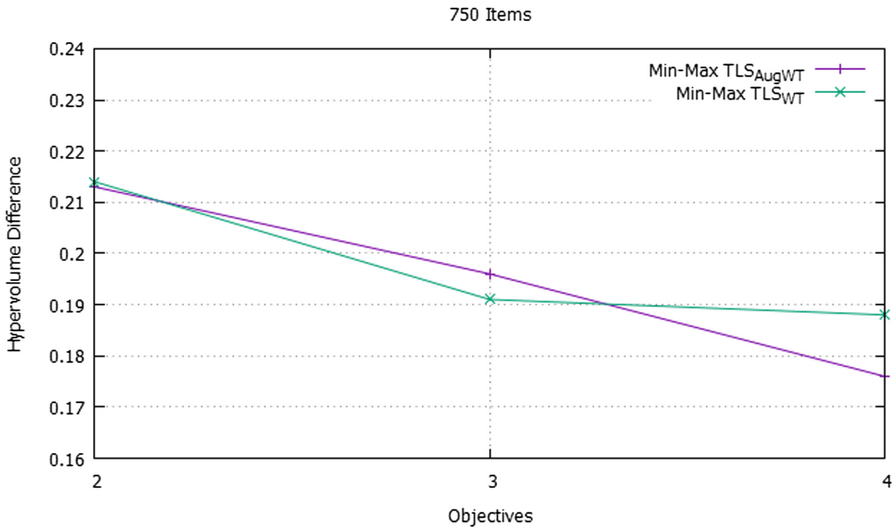


Fig. 4. Average hypervolume difference of *Min-Max* TLS$_{WT}$ and *Min-Max* TLS$_{AugWT}$ with 750 items [21].

Table 4. Average CPU Time of *Min-Max* TLS$_{WT}$, *Min-Max* TLS$_{AugWT}$, MOMAD, 2PPLS and MOTGA in seconds [21].

Instances	*Min-Max* TLS$_{WT}$	*Min-Max* TLS$_{AugWT}$	MOMAD	2PPLS	MOTGA
2×250	0.6	0.8	5.2	3.1	0.9
2×500	1.9	2.3	15.5	14.8	3.3
2×750	4.0	4.3	23.5	25.1	9.6
3×250	1.0	1.0	7.5	N/A	8.8
3×500	3.4	3.7	19.1	N/A	15.2
3×750	5.8	6.4	35.7	N/A	40.7
4×250	4.8	5.0	10.7	N/A	8.6
4×500	13.6	16.1	25.5	N/A	27.4
4×750	24.3	28.8	45.7	N/A	43.2

6.4 Discussion

As a conclusion of this experimentation section, the results have shown that the proposed approach performs better than the compared approaches. When evaluating the *Min-Max* TLS algorithm for solving the MOMKP, some useful conclusions can be extracted. First, the *Min-Max* TLS performs statistically better especially for the largest and hardest instances where the best values for the instances with 3 and 4 objectives are found by the two versions of *Min-Max* TLS. Second, the difference between the two versions *Min-Max* TLS$_{WT}$

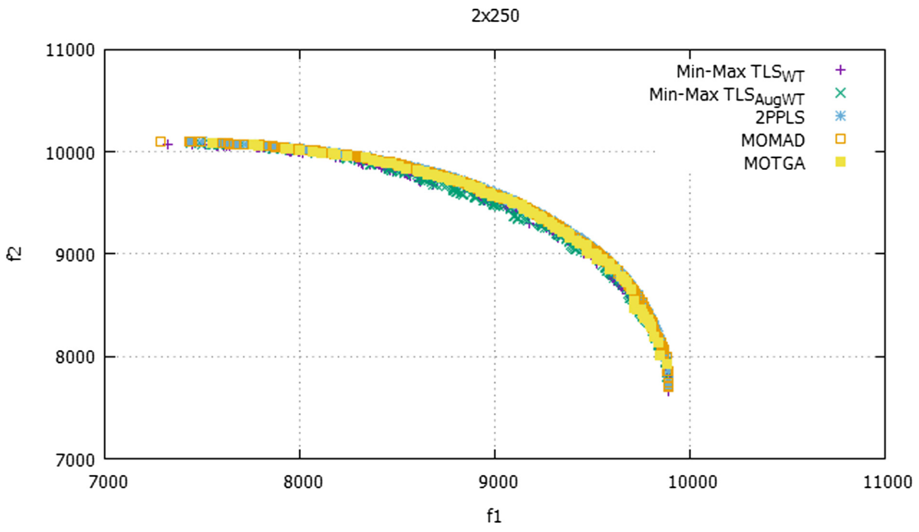


Fig. 5. Illustration of the median attainment surfaces obtained by *Min-Max* TLS$_{WT}$, *Min-Max* TLS$_{AugWT}$, MOMAD, 2PPLS, MOTGA with 2×250 instance [21].

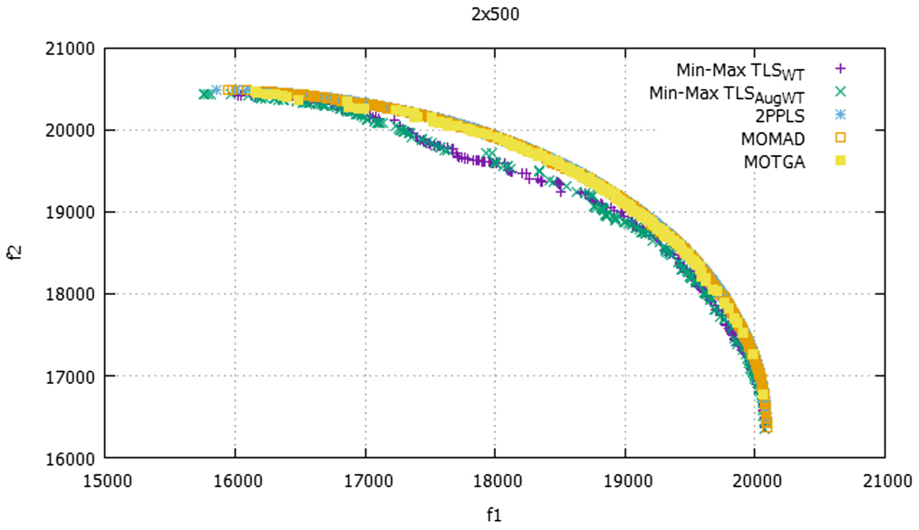


Fig. 6. Illustration of the median attainment surfaces obtained by *Min-Max* TLS$_{WT}$, *Min-Max* TLS$_{AugWT}$, MOMAD, 2PPLS, MOTGA with 2×500 instance [21].

and *Min-Max* TLS$_{AugWT}$ is very small. *Min-Max* TLS performs slightly better using *AugWT* function. Third, *Min-Max* TLS$_{AugWT}$ needs more computational time than *Min-Max* TLS$_{WT}$. In fact, the augmented weighted Tchebycheff version consumes slightly more CPU time than the weighted Tchebycheff version, especially with the largest instances but produces better results.

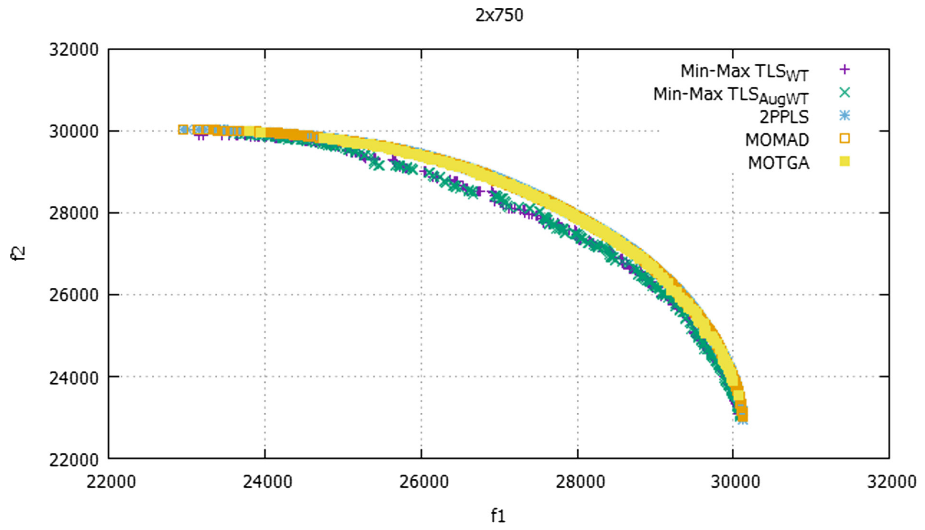


Fig. 7. Illustration of the median attainment surfaces obtained by *Min-Max* TLS$_{WT}$, *Min-Max* TLS$_{AugWT}$, MOMAD, 2PPLS, MOTGA with 2×750 instance [21].

Finally, let us mention that the efficiency of the proposed approach is due to many factors: the proposed Gw method, the perturbation function, the neighborhood structure, the acceptance criterion and the replacement strategy. The Gw method has a significant role in *Min-Max* TLS. Like any weighted approach, the selection of the weights can lead to a better performance. In fact, the Gw method tries to appropriately varying the search directions, all the member of population attempt to target almost all parts on the Pareto front. Hence, different Pareto-optimal points can be obtained. Furthermore, the initialization of the local search population is a very important function. The used perturbation function allows to generate a new population using information about the good solutions obtained during the previous iterations. Also, the neighborhood structure is a crucial part of the local search algorithm. Here, the proposed *Min-Max* $\mathcal{N}(s)$ tries to remove the least valuable item and replace it with the most profitable items according to the current solution. Therefore, it provides an efficient way to speed up the search while leading the algorithm to converge. Lastly, the acceptance criterion and the replacement strategy. The used replacement strategy in this work is a convergence-promoting mechanism, it favors exploitation. While the acceptance criterion, which can be considered as a diversity-promoting mechanism, favors more the exploration. Thus, both of them help to provide a good balance between exploration and exploitation during the search process.

7 Conclusions and Perspectives

This paper presents an approach to solve the multiobjective multidimensional knapsack problem. Based on local search method and the scalarization concept, more precisely the Tchebycheff metric, we propose two variants of Min-Max TLS: Min-Max TLS$_{WT}$ and Min-Max TLS$_{AugWT}$. Compared to three of the well-known state-of-the-art approaches, our experimental results have shown the efficiency of Min-Max TLS to solve almost all instances of MOMKP on a short processing time. Once we have different scalarizing functions, as improvement of this work, it would be interesting to propose an adaptive version of Min-Max TLS. In this version, an automatically mechanism can be implemented in order to choose between the Weighted Tchebycheff and the Augmented Weighted Tchebycheff, which can lead to better results since search ability depends strongly on these functions. Another perspective of this work is to combine Min-Max TLS with another metaheuristic method as the ant colony approach. The ant colony approach could replace the initial population function, since the use of this metaheuristic combined with a local search method is known to be efficient on many combinatorial problems.

Acknowlegment. This is an extended and revised version of a conference paper that was presented in ICSOFT 2017 [21].

References

1. Shih, H.: Fuzzy approach to multilevel knapsack problems. Comput. Math. Appl. **49**, 1157–1176 (2005)
2. Penn, M., Hasson, D., Avriel, M.: Solving the 0/1 proportional knapsack problem by sampling. J. Optim. Theory Appl. **80**, 261–272 (1994)
3. Smeraldi, F., Malacaria, P.: How to spend it: optimal investment for cyber security. In: Proceedings of the 1st International Workshop on Agents and CyberSecurity (2014)
4. Ehrgott, M., Ryan, D.M.: Constructing robust crew schedules with bicriteria optimization. J. Multi-Criteria Decis. Anal. **11**, 139–150 (2002)
5. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: improving the strength Pareto evolutionary algorithm for multiobjective optimization. In: Giannakoglou, K., et al. (eds.) Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001), International Center for Numerical Methods in Engineering (CIMNE), vol. 1, pp. 95–100 (2002)
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**, 181–197 (2002)
7. Lust, T., Teghem, J.: Memots: a memetic algorithm integrating tabu search for combinatorial multiobjective optimization. RAIRO - Oper. Res. **42**, 3–33 (2008)
8. Alaya, I., Solnon, C., Ghédira, K.: Ant colony optimization for multi-objective optimization problems. In: 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), vol. 1, pp. 450–457 (2007)
9. Ben Mansour, I., Alaya, I.: Indicator based ant colony optimization for multiobjective knapsack problem. In: 19th Annual Conference Knowledge-Based and Intelligent Information & Engineering Systems, vol. 60, pp. 448–457 (2015)
10. Alsheddy, A., Tsang, E.: Guided Pareto local search and its application to the 0/1 multi-objective knapsack problems. In: Proceedings of the Eighth Metaheuristic International Conference (MIC 2009) (2009)

11. Vianna, D.S., Dianin, M.F.: Local search based heuristics for the multiobjective multidimensional knapsack problem. Prod. J. **1**, 478–487 (2013)
12. Liefooghe, A., Paquete, L., Figueira, J.: On local search for bi-objective knapsack problems. Evol. Comput. **21**, 179–196 (2013)
13. Ehrgott, M., Gandibleux, X.: Approximative solution methods for multiobjective combinatorial optimization. Top **12**, 1–63 (2004)
14. Lust, T., Teghem, J.: The multiobjective multidimensional knapsack problem: a survey and a new approach. Int. Trans. Oper. Res. **19**, 495–520 (2012)
15. Alves, M.J., Almeida, M.: MOTGA: a multiobjective Tchebycheff based genetic algorithm for the multidimensional knapsack problem. Comput. OR **34**, 3458–3470 (2007)
16. Ke, L., Zhang, Q., Battiti, R.: A simple yet efficient multiobjective combinatorial optimization method using decomposition and Pareto local search. IEEE Trans. Cybern. (2014)
17. Zhang, Q., Li, H.: MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans. Evol. Comput. **11**, 712–731 (2007)
18. Bowman, V.J.: On the relationship of the Tchebycheff norm and the efficient frontier of multiple-criteria objectives. In: Thieriez, H., Zionts, S. (eds.) Multiple Criteria Decision Making, vo. 1, pp. 76–85 (1976)
19. Steuer, R.E., Choo, E.U.: An interactive weighted Tchebycheff procedure for multiple objective programming. Math. Program. **26**(3), 326–344 (1983)
20. Steuer, R.E.: Multiple Criteria Optimization: Theory, Computation and Application. Wiley, New York (1986)
21. Ben Mansour, I., Alaya, I., Tagina, M.: A min-max Tchebycheff based local search approach for MOMKP. In: Proceedings of the 12th International Conference on Software Technologies, ICSOFT, INSTICC, vol. 1, pp. 140–150. SciTePress (2017)
22. Alaya, I., Solnon, C., Ghédira, K.: Ant algorithm for the multi-dimensional knapsack problem. Proc. Int. Conf. Bioinspir. Optim. Methods Appl. (BIOMA) **1**, 63–72 (2004)
23. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans. Evol. Comput. **1**, 257–271 (1999)
24. Knowles, J.D., Thiele, L., Zitzler, E.: A tutorial on the performance assessment of stochastive multiobjective optimizers. Technical report TIK-Report (2005)
25. Grunert da Fonseca, V., Fonseca, C.M., Hall, A.O.: Inferential performance assessment of stochastic optimisers and the attainment function. In: Zitzler, E., Thiele, L., Deb, K., Coello Coello, C.A., Corne, D. (eds.) EMO 2001. LNCS, vol. 1993, pp. 213–225. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44719-9_15

Monitoring and Control of Vehicles' Carbon Emissions

Tsvetan Tsokov[1] and Dessislava Petrova-Antonova[2](✉)

[1] Department of Information Technologies, Sofia University, 5 James Bourchier Blvd., 1164 Sofia, Bulgaria
tsvetan.tso@gmail.com

[2] Department of Software Engineering, Sofia University, 125 Tsarigradsko shoes Blvd., 1113 Sofia, Bulgaria
d.petrova@fmi.uni-sofia.bg

Abstract. Machine-to-machine communication, known as Internet of Things (IoT), allows machines to connect using variety of sensors and devices. Thus, feedbacks to govern energy, agriculture, transportation and environment are able to be obtained. The IoT provides opportunity not only for creation of new businesses and investments, but for reduction of carbon emissions. It enables production of highly automated and connected vehicles that change the global automotive market. Following the current IoT trends, this paper proposes a solution for real-time monitoring of vehicles and control of carbon emissions, called EcoLogic. The EcoLogic is composed of hardware module and several applications providing cloud services. The hardware module is based on Arduino and Raspberry Pi embedded systems and measures several physical parameters by sensors or extracts them from the onboard diagnostic system of the vehicle for further analysis. The cloud applications process the incoming data, store it into a SAP HANA database and analyze it to provide feedback and control of carbon emissions.

Keywords: Clustering · Internet of Things · Reduction of carbon emissions
Sensor data processing

1 Introduction

1.1 A Subsection Sample

As of now, in Europe we witness the outburst of a new industrial revolution, labelled Industry 4.0 and driven by new-generation ICT such as Big Data, Internet of Things (IoT), Cloud computing, Internet of Services (IoS), Robotics, etc. The ubiquitous use of sensors, wide spread of wireless communications and networks, growing computing power at lower cost and the development of 'Big Data' analytics, as well as the deployment of increasingly intelligent robots and machines, has the potential to transform the way goods are manufactured in Europe [5]. Industry as one of the pillars of the EU economy – the manufacturing sector accounts for 2 million enterprises, 33 million jobs and 60% of productivity growth. The new, digital industrial revolution provides the businesses with technologies for bigger flexibility, extensive customization, increased speed, better quality and improved productivity. It opens new horizons for enterprises to become more adventurous, to develop innovative products and services which better serve the needs of the customers.

According to Gartner, the IoT is “the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment”. As sensors spread across almost every industry, the IoT triggers a massive influx of Big Data and is one of the key drivers of the Big Data phenomenon. According to EC the market value of the IoT in the EU is expected to exceed one trillion euros in 2020. For the past six years, the EC has been actively cooperating with the industry and with EU Member States and third countries to unleash the potential of the IoT technology. Nevertheless, the huge variety of vendors and the low level of standardization create severe problems of interoperability. Moreover, as the number of devices increases drastically, this poses additional challenges related to security especially in the case of heterogeneity and limitations of network capacities.

One of the sectors that is most affected by the IoT is automotive industry. IoT technologies enable production of highly automated and connected vehicles that will change the global automotive market. Recently, tens of millions of cars are said to be connected to the Internet and their number is expected to become hundreds of millions in the near future [6]. At the same time, mobile communication technology is recognized to have considerable potential to enable carbon emissions reduction across a variety of applications in a wide range of sectors [7]. According to Global e-Sustainability Initiative, 70% of the carbon savings currently being made come from the use of machine-to-machine (M2 M) technologies. The greater savings comes from buildings (29%) and transportation (28%). The survey data shows that 68% of smartphone users are willing to adopt behaviours that could result in even more substantial future reductions to personal carbon emissions. IoT is pointed as a key lever to reduce the carbon emissions in a statistic of A.T. Kearney [8]. In particular, car sharing, automotive telematics and smart home are the most promising cases.

Inspired by the low-carbon economy roadmap of European commission and the grate opportunity provided by the IoT technologies for reducing the carbon emissions, this paper proposes a solution for real-time monitoring of vehicles and detection of rising levels of carbon emissions, called EcoLogic. The proposed solution includes hardware module, which collects sensor data related to vehicle’s carbon emissions such as air pressure, air temperature and fuel mixture and sends it to cloud-based applications for further analysis. The results from the analysis are used to control the carbon emissions through smart notifications and vehicle’s power limitations. A preliminary implementation of EcoLogic platform is outlined in [9]. The contribution of this paper consists in:

- more comprehensive description EcoLogic platform’s architecture and implementation;
- an empirical evidence of the feasibility of EcoLogic platform through usage example from real environment.

The rest of the paper is organized as follows. Section 2 outlines the related work. Section 3 presents the general concept of EcoLogic platform, while Sect. 4 describes its architecture. Section 5 the usage of EcoLogic platform in real environment. Finally, Sect. 6 concludes the paper and gives directions for future work.

2 Related Work

The current software solutions for tracking and monitoring vehicles give evidence for the efforts of using IoT technologies in automotive industry. This section presents those that are closest in functionality to the proposed one.

Geotab provides a service for monitoring and analysis of vehicles using integrated hardware module that sends data to private cloud platform. The hardware module is directly connected to the onboard diagnostic system of the vehicle and collects data about fuel consumption, travelled distance and other parameters [1]. The analysis provided by the cloud platform allows identification of vehicles with suboptimal fuel consumption [2]. Unfortunately, Geotab solution does not provide control over vehicle's parameters and does not detect anomalies related to increasing rate of carbon emissions.

The data logger of Madgetech provides functionality for regular monitoring of carbon dioxide levels [3]. It measures the carbon emissions in exhaust system of vehicles and sends data to private cloud platform through a wireless network. The measured data is visualized by mobile application, but further analysis is not supported. In addition, functionality for control of the carbon emissions is not provided.

The CanTrack solution provides a system for real-time GPS tracking of vehicles [4]. Its Driver Behaviour Module support driver profiling based on 5 key driving elements including driving style, speeding and idling. The drivers are assisted to avoid traffic delays, blocked roads and accidents through real-time and directional traffic information. After a collision has been detected system alerts are generated in order to provide accurate location information to emergency services if required. A drawback of the CanTrack solution is that it works only with GPS data and does not takes into account the vehicle's parameters related to carbon emissions.

The presented software solutions for tracking and monitoring vehicles are compared using the following criteria:

- C1: Measurement of large range of parameters through sensors, including direct or indirect measurement of carbon emissions' rate.
- C2: Sending sensor data to cloud platform for storage.
- C3: Detection of anomalies related to increasing rate of carbon emissions or catching of vehicles' failures.
- C4: Support of mechanism for notification and control over vehicle's parameters using effectors.
- C5: Support of configuration allowing system to work with heterogeneous vehicles, sensors and effectors.
- C6: Providing of Software Development Kit.

The results from comparison are presented in Table 1.

Table 1. Comparison of software solutions for tracking and monitoring vehicles.

Criterion	Geotab	Madgetech	Cantrack
C1	Yes	No	No
C2	Yes	Yes	Yes
C3	No	No	No
C4	No	No	Yes
C5	Yes	No	No
C6	Yes	No	No

As can be seen from Table 1, all software solutions store the collected vehicles' data on a cloud platform. Unfortunately, they use private cloud and thus the data is not accessible for analysis by the stakeholders. Furthermore, none of them perform detection of anomalies related to increasing rate of carbon emissions or catching of vehicles' failures. Support of mechanism for notification and control over vehicle's parameters using effectors is not provided by Geotab and Madgetech. Therefore, to the best of our knowledge there is no complete solution that covers all functional characteristics related to the comparison criteria above.

3 EcoLogic General Concept

The EcoLogic is composed of hardware modules, which are installed on vehicles and applications providing services, which are deployed on a cloud platform. Its architecture is shown in Fig. 1.

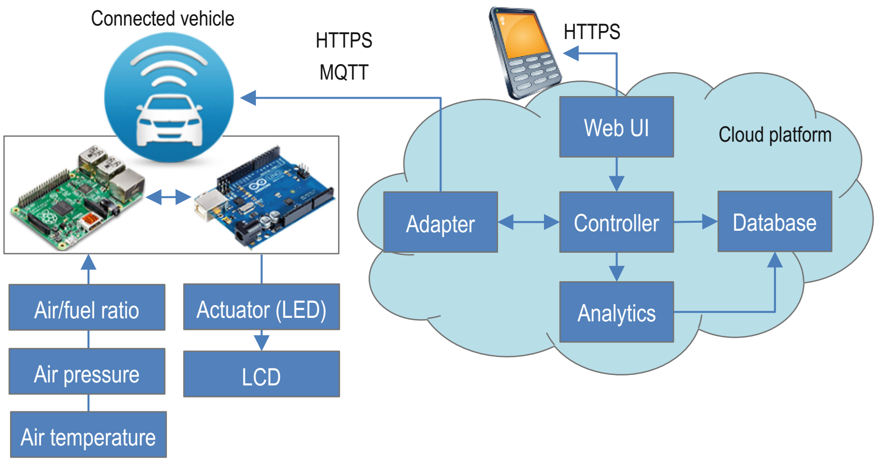


Fig. 1. EcoLogic general architecture.

The hardware module measures several physical parameters by sensors or extracts them from the onboard diagnostic system of the vehicle. The data is sent to the cloud platform. The measured physical parameters are:

- Air/fuel ratio, which is measured by lambda sonde sensor, which is located into the exhaust system of the vehicle.
- Absolute pressure of the air that is consumed by the engine.
- Temperature of the air that is consumed by the engine.

The cloud applications are implemented as microservices, which are designed in a platform independent way in order to have the possibility for deployment on different cloud platforms. The cloud applications are communicating with a database, which is provided by backing service from the cloud platform. They process the incoming data, store it into the database and analyze it. The hardware modules communicate with the cloud platform with wireless network via HTTPS or MQTT protocols. The following physical parameters are calculated on the base of the incoming sensor data:

- Mass of the consumed air by the engine;
- Mass of the consumed fuel by the engine;
- Mass of the carbon dioxide emissions, exposed into the atmosphere.

All measured and calculated physical parameters are stored in the database. A cloud-based *Analytics* application performs an anomaly detection on the streamed data by searching for vehicles that have not optimal amount of carbon dioxide emissions or system failures. The anomaly detection process is based on clustering analysis. When some vehicle is detected by the system as an anomaly, with non-optimal amount of emissions, the hardware module is notified automatically by the cloud platform and hardware actuator is activated to reduce the amount of emissions. In this way the system monitors and controls the amount of carbon dioxide emissions in the atmosphere in real time. The hardware modules are equipped with three actuators:

- Liquid crystal display (LCD), which visualize the measured and calculated physical parameters to the driver.
- Light-emitting diode (LED), which indicates to the driver that the amount of carbon dioxide emissions is not optimal or there is a system failure (not optimal parameters).
- Actuator, which controls the amount of injected fuel in the engine and regulates the amount of emissions.

Currently, the EcoLogic has only the LCD display and LED. The purpose of the LED is to notify the driver to manually reduce the speed and change the driving behaviour, which leads to reduction of the amount of emissions.

The cloud platform provides web user interface that is publicly available and accessible by clients via HTTPS protocol.

The user management of the system is composed of two roles: driver and operator. The process flow of the system is the following:

- Driver buys a hardware module from a dealer.
- The driver installs the hardware module into vehicle.
- The driver registers the vehicle with the hardware module and sensors in the system. All components have unique identification numbers.
- Drivers are authorized to monitor, analyze and control their own registered vehicles.
- Operators are authorized to monitor, analyze and control all registered vehicles by regions.
- Each driver gets score points proportional to the amount of carbon dioxide emissions exposed in the atmosphere by their vehicles. Drivers can participate in greenhouse gas trading and decrease pollution taxes with their score points.

4 EcoLogic Architecture and Implementation

The EcoLogic platform consists of six software components: *Arduino* application, *Raspberry Pi* application, *Java EE* backend application, *JavaScript Web UI* application, *Analytics* application and SAP HANA Database. Its software architecture is presented in Fig. 2.

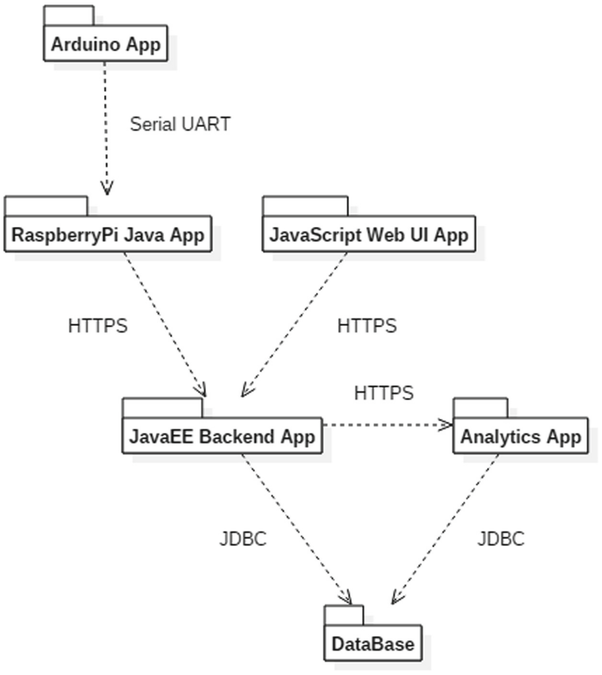


Fig. 2. EcoLogic software architecture.

The *Arduino* and *Raspberry Pi* applications work on *Olimexino328* and *Raspberry Pi B+* embedded systems accordingly. They built the hardware module that operate on the vehicle. The Java EE Backend application is a server-side application working on SAP Cloud Platform. It provides REST web services to the *Raspberry Pi* application and *JavaScript Web UI* application using HTTPS protocol. The *Raspberry Pi* application communicates with the *Java EE backend* application, which performs processing of the measured parameters from the hardware module and sends notifications for control of the effector in the vehicle. The *Java EE backend* application is connected to the *Database*, where the vehicles' parameters and rules for anomaly detection are stored. The *Analytics* application performs data analysis using queries and stored procedures provided by the database. It provides REST web services to the *Java EE Backend* application over HTTPS protocol.

4.1 EcoLogic Database

The SAP NAHA Database of EcoLogic platform supports Multitenant Database Containers (MDCs). The MDC system always has one system database and zero or several MDCs, also known as tenant databases. Two database schemas are created for the purpose of EcoLogic platform. They are shown in presented in Fig. 3. The first one, called MAIN schema, includes three tables, namely *Vehicle*, *Sensor* and *Measurement*. The second one, named MDC_DATA schema, is created for the purpose of analysis. It has one table, named *Measurements*, that is created in the context of the *Vehicle* table.

The table *Vehicle* of the MAIN schema stores data for the vehicles such as model, engine capacity of the vehicle, mass of the consumed air by the engine, mass of the consumed fuel by the engine, mass of the carbon dioxide emissions, ecological state of the vehicle and limit of the carbon dioxide emissions. Additional column is added to

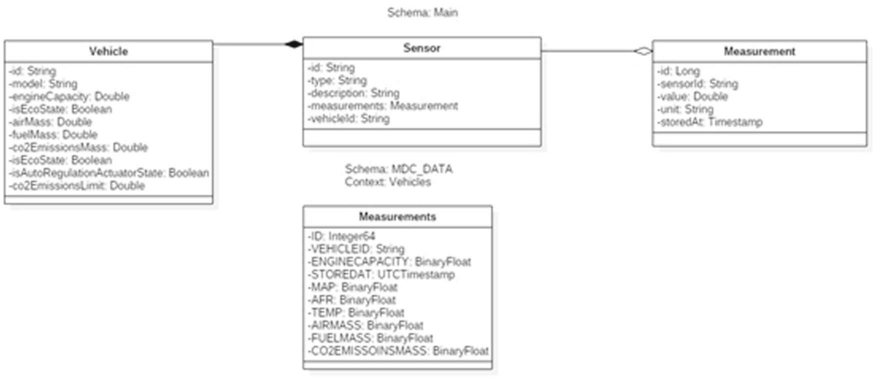


Fig. 3. Database model of EcoLogic platform.

store Boolean value indicating if the state of the Actuator should be regulated automatically or not. The primary key of the *Vehicle* table uniquely identifies each vehicle's record in it.

The *Sensor* table of the MAIN schema stores data for the sensors such as type of the sensor, description, vehicle's identifier and measurements. The relationship between the *Vehicle* table and *Sensor* table is one to many. Thus, the physical parameters of each vehicle could be measured with different type of sensors. Currently, three types of sensors are supported: MAP for measuring the pressure of the air, TEMP for measuring the temperature of the air and AFR for measuring the air/fuel ratio. The primary key of the *Sensor* table uniquely identifies each sensor's record in it.

The *Measurement* table of the MAIN schema stores data for the measured values by the sensors such as unit, value and timestamp. The relationship between the *Sensor* table and *Measurement* table is one to many. Thus, each sensor could provide zero or more measurements of particular physical parameter.

The *Measurements* table of the MDC_DATA schema gives a view of the monitored vehicles and their last measured physical parameters. It is used to perform clustering analysis by the *Analytics* application.

4.2 EcoLogic Hardware Module

As it was already mentioned, the EcoLogic hardware module consists of *Olimexino328* and *Raspberry Pi B+* embedded systems, communicating each other. Its architecture is presented in Fig. 4.

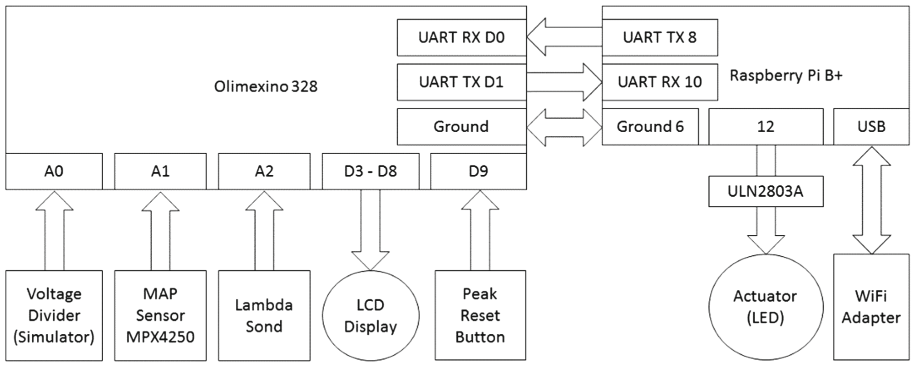


Fig. 4. Architecture of the hardware module.

The *Arduino* application, working on the *Olimexino328* embedded system, is implemented with the programming language C++ using Wiring library. It measures the physical parameters of the vehicle through the sensors, processes and visualizes them on the 4 × 16 LCD display.

The *Raspberry Pi* application acts as a proxy between the *Arduino* application and the *JavaScript Web UI* application. It receives the measured physical parameters from the *Arduino* application and sends them to the *JavaScript Web UI* application. It receives notifications in case of non-optimal physical parameters or rising levels of carbon dioxide emissions. The *Raspberry Pi B+* embedded system uses a system on a chip (SoC), which includes an ARM compatible central processing unit and Raspbian operational system that is Linux based. It is connected to the internet through 802.11n wireless network. The *Raspberry* application is implemented with Java programming language using WiringPi, Pi4 J and Apache HttpClient libraries.

4.3 Java EE Backend Application

The *Java EE backend* application is implemented with Java Enterprise Edition. It is a server-side application that provides web services implemented as Java servlets: *AddMeasurementServlet*, *DataManagerServlet* and *AnalyticsServlet*. Its class diagram is shown in Fig. 5.

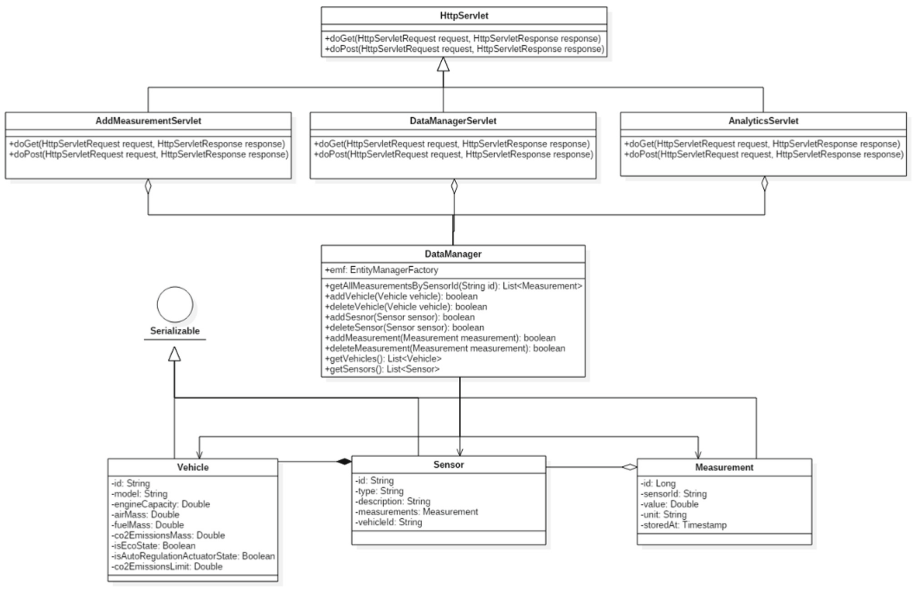

HttpServlet
+doGet(HttpServletRequest request, HttpServletResponse response)
+doPost(HttpServletRequest request, HttpServletResponse response)
AddMeasurementServlet
DataManagerServlet
AnalyticsServlet
DataManager
+emf: EntityManagerFactory
+getAllMeasurementsBySensorId(String id): List<Measurement>
+addVehicle(Vehicle vehicle): boolean
+deleteVehicle(Vehicle vehicle): boolean
+addSesnor(Sensor sensor): boolean
+deleteSensor(Sensor sensor): boolean
+addMeasurement(Measurement measurement): boolean
+deleteMeasurement(Measurement measurement): boolean
+getVehicles(): List<Vehicle>
+getSensors(): List<Sensor>
Serializable
Vehicle
-id: String
-model: String
-engineCapacity: Double
-airMass: Double
-fuelMass: Double
-co2EmissionsMass: Double
-isEcoState: Boolean
-isAutoRegulationActuatorState: Boolean
-co2EmissionsLimit: Double
Sensor
-id: String
-type: String
-description: String
-measurements: Measurement
-vehicleId: String
Measurement
-id: Long
-sensorId: String
-value: Double
-unit: String
-storedAt: Timestamp

Fig. 5. Class diagram of *Java EE backend* application.

The *AddMeasurementServlet* servlets communicates with the hardware module. It receives data, interprets, processes and stores it to the MAIN:Measurement and MDC_DATA.Vehicles:Measurements tables of the database. The *DataManagerServlet* servlet performs create, read, update and delete (CRUD) operations over Vehicle, Sensor and Management tables of the database for the purpose of the *JavaScript Web UI* application. The *AnalyticsServlet* servlet communicates with the *Analytics* application in order to provide JSON data related to the detected anomalies.

4.4 Analytics Application

The *Analytics* application consists of the following components:

- Java Analytics Application;
- Automated Predictive Analytics Library that provides machine learning algorithms;
- Measurements Application that provides web services for create, read and update of data.

The *Analytics* application operates on the *Measurements* table of MDC_DATA database schema. It uses K-Means algorithm for clustering analysis, where the number of clusters (K) is the number of engine capacities of the registered vehicles in the EcoLogic platform. The *Analytics* application provides REST API that handles requests and responses in JSON format. Listing 1 shows the request for anomaly detection, while Listing 2 presents the corresponding response.

Listing 1. Request for anomaly detection.

```
Host: /api/analytics/outliers/sync
Method: HTTP GET
Body (JSON):
{
  "datasetID": 1,
  "targetColumn": "CO2EMISSIONSMASS",
  "skippedVariables": ["ID", "STOREDAT", "MAP", "AFR", "TEMP"],
  "numberOfOutliers": 10,
  "weightVariable":"AIRMASS",
  "numberOfReasons": 4
}
```

Listing 2. Response of anomaly detection

```
{
"modelPerformance": {
    "confidenceIndicator": 1,
    "predictionConfidence": 0.9942,
    "predictivePower": 0.995,
    "qualityRating": 5
  },
  "numberOfOutliers": 1,
  "outliers": [
    {
      "dataPoint": {
        "VEHICLEID": "7252436",
        "ENGINECAPACITY": 1400,
        "AIRMASS": 25.57,
        "FUELMASS": 3.48,
        "CO2EMISSIONSMASS": 6.02
      },
      "errorBar": 0.2220971039702342,
      "predictedValue": 5.733528320778292,
      "realValue": "6.02",
      "reasons": [
        {
          "value": "3.48",
          "variable": "FUELMASS"
        },
        {
          "value": "25.57",
          "variable": "AIRMASS"
        },
        {
          "value": "1400",
          "variable": "ENGINECAPACITY"
        }     ]}]}
```

4.5 JavaScript Web UI Application

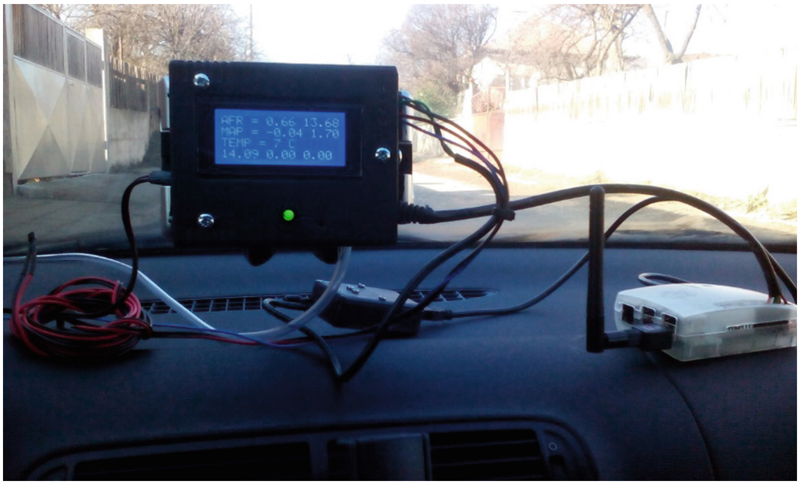
The *JavaScript Web UI* application is implemented with ECMAScript 5, HTML5, CSS and SAPUI5. It follows the Model-View-Controller (MVC) architecture and provides the following functionality:

- Performs create, read, update and delete (CRUD) operations over the database;
- Visualizes the current physical parameters of the vehicles;
- Provides hand control of the vehicles' emissions;
- Activation and deactivation of the option for automatic control of the vehicles' emissions;
- Visualizes the vehicles with non-optimal physical parameters or rising levels of carbon dioxide emissions.

5 EcoLogic Usage Example

This section provides evidence for functioning of the EcoLogic platform in real environment. The user interface of both hardware and software modules is presented.

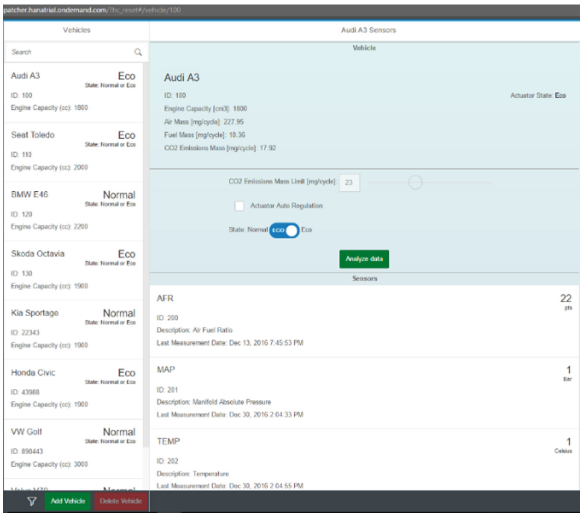
Figure 6 shows the hardware module installed on a real vehicle. It is equipped with the *Olimexino328* and *Raspberry Pi B+* embedded systems, power box and wireless adapter. The LCD display visualizes the pressure and the temperature of the air, as well as the air/fuel ration.



```
AFR = 0.66 13.68
MAP = -0.04 1.70
TEMP = 7 C
14.09 0.00 0.00
```

Fig. 6. User interface of hardware module.

Figure 7 shows a dialog from the web user interface, where the automatic control of the emissions is switched off. The data on the screen is presented on two panels, named *Vehicle* and *Sensors*.



patcher.hanatrial.ondemand.com/?hc_reset#/vehicle/100

Audi A3 Eco
State: Normal or Eco
ID: 100
Engine Capacity (cc): 1800

Seat Toledo Eco
State: Normal or Eco
ID: 110
Engine Capacity (cc): 2000

BMW E46 Normal
State: Normal or Eco
ID: 120
Engine Capacity (cc): 2200

Skoda Octavia Eco
State: Normal or Eco
ID: 130
Engine Capacity (cc): 1900

Kia Sportage Normal
State: Normal or Eco
ID: 22343
Engine Capacity (cc): 1900

Honda Civic Eco
State: Normal or Eco
ID: 43988
Engine Capacity (cc): 1900

VW Golf Normal
State: Normal or Eco
ID: 890443
Engine Capacity (cc): 3000

Audi A3
ID: 100
Actuator State: Eco
Engine Capacity [cm3]: 1800
Air Mass [mg/cycle]: 227.95
Fuel Mass [mg/cycle]: 10.36
CO2 Emissions Mass [mg/cycle]: 17.92

CO2 Emissions Mass Limit [mg/cycle]: 23
Actuator Auto Regulation
State: Normal ECO Eco

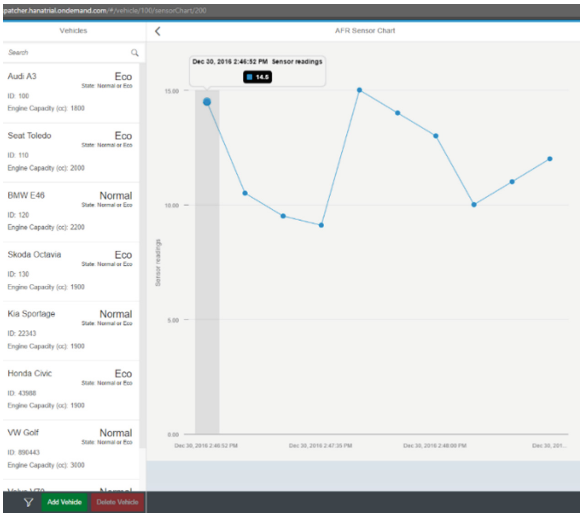
AFR 22 pts
ID: 200
Description: Air Fuel Ratio
Last Measurement Date: Dec 13, 2016 7:45:53 PM

MAP 1 Bar
ID: 201
Description: Manifold Absolute Pressure
Last Measurement Date: Dec 30, 2016 2:04:33 PM

TEMP 1 Celsius
ID: 202
Description: Temperature
Last Measurement Date: Dec 30, 2016 2:04:55 PM

Fig. 7. User dialog with vehicle data.

The *Vehicle* panel shows the data for the selected vehicle such as engine capacity, air and fuel mass, carbon dioxide emissions, etc. The *Sensors* panel displays the last collected data from sensors such as temperature and pressure of the air and air/fuel ratio. When the user selects a particular physical parameter, a new dialog with the latter measurements from the corresponding sensor is shown. It is presented in Fig. 8.



patcher.hanatrial.ondemand.com/#/vehicle/100/sensorChart/200

Audi A3 Eco
State: Normal or Eco
ID: 100
Engine Capacity (cc): 1800

Seat Toledo Eco
State: Normal or Eco
ID: 110
Engine Capacity (cc): 2000

BMW E46 Normal
State: Normal or Eco
ID: 120
Engine Capacity (cc): 2200

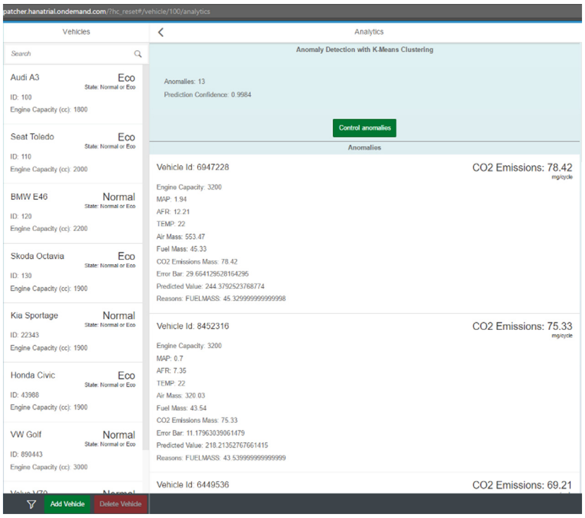
Skoda Octavia Eco
State: Normal or Eco
ID: 130
Engine Capacity (cc): 1900

Kia Sportage Normal
State: Normal or Eco
ID: 22343
Engine Capacity (cc): 1900

Honda Civic Eco
State: Normal or Eco
ID: 43988
Engine Capacity (cc): 1900

VW Golf Normal
State: Normal or Eco
ID: 890443
Engine Capacity (cc): 3000

Fig. 8. Latter measurements of given sensor.



Audi A3 Eco
State: Normal or Eco
ID: 100
Engine Capacity (cc): 1800

Seat Toledo Eco
State: Normal or Eco
ID: 110
Engine Capacity (cc): 2000

BMW E46 Normal
State: Normal or Eco
ID: 120
Engine Capacity (cc): 2200

Skoda Octavia Eco
State: Normal or Eco
ID: 130
Engine Capacity (cc): 1900

Kia Sportage Normal
State: Normal or Eco
ID: 22343
Engine Capacity (cc): 1900

Honda Civic Eco
State: Normal or Eco
ID: 43988
Engine Capacity (cc): 1900

VW Golf Normal
State: Normal or Eco
ID: 890443
Engine Capacity (cc): 3000

Anomaly Detection with K-Means Clustering

Anomalies: 13
Prediction Confidence: 0.9984

Anomalies

Vehicle Id: 6947228 CO2 Emissions: 78.42 mg/cycle
Engine Capacity: 3200
MAP: 1.94
AFR: 12.21
TEMP: 22
Air Mass: 553.47
Fuel Mass: 45.33
CO2 Emissions Mass: 78.42
Error Bar: 29.664129528164295
Predicted Value: 244.3792523768774
Reasons: FUELMASS: 45.329999999999998

Vehicle Id: 8452316 CO2 Emissions: 75.33 mg/cycle
Engine Capacity: 3200
MAP: 0.7
AFR: 7.35
TEMP: 22
Air Mass: 320.03
Fuel Mass: 43.54
CO2 Emissions Mass: 75.33
Error Bar: 11.17963039061479
Predicted Value: 218.21352767661415
Reasons: FUELMASS: 43.539999999999999

Vehicle Id: 6449536 CO2 Emissions: 69.21

Fig. 9. Detection of anomalies.

When the user clicks on the "Analyze data" button of the dialog from Fig. 7, a new dialog with the results after execution of the algorithm for anomaly detection is coming up. It is presented in Fig. 9.

The dialog in Fig. 9 has two panels, named *Anomaly Detection with K-means clustering* and *Anomalies*. The first one shows the parameters for execution the the K-means clustering algorithm, while the second one presents a list with vehicles for which anomalies are found. Except the physical parameters of the vehicles, three additional parameters can be seen as follows:

- Predicted value – calculated expected value of carbon dioxide emissions;
- Error bar – magnitude of the error that is calculated based on the expected and obtained values of the carbon dioxide emissions;
- Reasons – parameters that indicate anomaly.

The results from testing the analytics functionality of EcoLogic platform are presented in [9]. A case study with two datasets was performed. The first data set contains known anomalies and is publicly available [10]. The second one collects data from a real vehicle with internal combustion engine, which works on petrol and has capacity of 1800 cubic centimeters. It was extended proportionally with appropriate simulated data in order to obtain bigger dataset. The obtained results from the experiments prove the feasibility of the EcoLogic platform to detect anomalies in the vehicles' behavior related to increased carbon emissions.

6 Conclusions and Future Work

The present work addresses the applicability of IoT technologies to reduce the carbon emissions produced by the vehicles. It presents a fully completed solution called EcoLogic, which is ready for production usage to solve a global problem for the environment. The main advantage of EcoLogic platform is that it is independent from the underlying hardware and software due to ability to work with variety sensors or extract data from the onboard diagnostic system of different vehicles as well as ability to deploy on diverse cloud environments. In addition, cloud implementation based on micro-services provides high scalability, resilience and opportunity to work with big amounts of data.

Future work includes integration of new application protocols such as CoAP, DDS and AMQP that can be used by default, since currently only HTTPS and MQTT are supported. Implementation of analytics functionality for prediction of potential failures in vehicles, based on the current and historical data is also considered.

Acknowledgements. The authors acknowledge the financial support by the National Scientific Fund, Bulgarian Ministry of Education and Science within the project no. DN02/11 and project no. DN12/9, and by the Scientific Fund of Sofia University within project no. 80-10-162/25.04.2018.

References

1. GO7. UK: Geotab (2017)
2. MyGeotab. UK: Geotab (2017)
3. Data Loggers. USA: Madgetech
4. CanTrack GPS. UK: CanTrack Global Ltd
5. Davies, R.: Industry 4.0 digitalisation for productivity and growth. European Union (2015)
6. Automotive IT-Kongress 4.0 (2015): http://www.t-systems.de/news-media/automotiveit-kongress-industrie-4-0-veraendert-automobilindustrie/1339486. Accessed 13 Nov 2017
7. Stephens, A., Iglesias, M., Plotnek, J.: GeSI Mobile Carbon Impact (2015). http://gesi.org/files/Reports/GeSI%20mobile%20carbon%20impact.pdf. Accessed 13 Nov 2017
8. A.T. Kearney: Internet of Things, a key lever to reduce CO2 emissions (2015). http://www.atkearney.fr/documents/877508/879237/20151113_IoT+Impact+on+energy_Europe+EN.pdf/6757111f-21da-49ee-82fd-915ff42dc26d. Accessed 12 Nov 2017
9. Tsokov, T., Petrova-Antonova, D.: EcoLogic: IoT platform for control of carbon emissions. In: Proceeding of the 12th International Conference on Software Technologies (ICSOFT), Madrid, Spain, 24–26 July, vol. 1, pp. 178–185 (2017)
10. Philip Mugglestone: SAP HANA Academy (2014). https://github.com/saphanaacademy/PAL/tree/master/Source%20Data/PAL. Accessed 06 Feb 2017

WOF: Towards Behavior Analysis and Representation of Emotions in Adaptive Systems

Ilham Alloui(✉) and Flavien Vernier

LISTIC, Univ. Savoie Mont Blanc, 74000 Annecy, France
ilham.alloui@univ-smb.fr

Abstract. With the increasing use of new technologies such as Communicating Objects (COT) and the Internet of Things (IoT) in our daily life (connected objects, mobile devices, etc.), designing Intelligent Adaptive Distributed software Systems (DIASs) has become an important research issue. Human face the problem of mastering the complexity and sophistication of such systems as those require an important cognitive load for end-users who usually are not expert. Starting from the principle that it is to technology-based systems to adapt to end-users and not the reverse, we address the issue of how to help developers design and produce such systems. We then propose WOF, an object oriented Framework founded on the concept of *Wise Object* (WO), a metaphor to refer to human introspection and learning capabilities.

To make systems able to learn by themselves, we designed introspection, monitoring and analysis software mechanisms such that WOs can learn and construct their own knowledge. We then define a WO as a software-based entity able to learn by itself on itself (i.e. on services it is intended to provide) and also on the others (i.e. the way others use its services). A WO is seen as an avatar of either a physical or a logical object (e.g. device/software component).

In this paper, we introduce the main requirements for DIASs as well as the design principles of WOF. We detail the WOF conceptual architecture and the Java implementation we built for it. To provide application developers with relevant support, we designed WOF with the minimum intrusion in the application source code. Adaptation and distribution related mechanisms defined in WOF can be inherited by application classes. In our Java implementation of WOF, object classes produced by a developer inherit the behavior of *Wise Object* (WO) class. An instantiated system is a *Wise Object System* (WOS) composed of WOs that interact through an event bus. In the first version of WOF, a WO was able to use introspection and monitoring built-in mechanisms to construct knowledge on: (a) services it is intended to render; (b) the usage done of its services. In the current version, we integrated an event-based WO simulator and a set of Analyzer classes to provide a WO with the possibility to use different analysis models and methods on its data. Our major goal is that a WO can be able to identify common usage of its services and to detect unusual usage. We use the metaphor of emotions to

© Springer International Publishing AG, part of Springer Nature 2018
E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 244–267, 2018.
https://doi.org/10.1007/978-3-319-93641-3_12

refer to unusual behavior (stress, surprise, etc.). We show in the paper a first experiment based on a statistical analysis method founded on stationary processes to identify usual/unusual behavior.

Keywords: Object oriented design · Architecture models
Adaptive systems · Introspection · Decentralized control
Behavior analysis · Emotion representation

1 Introduction

With the increasing use of new technologies such as Communicating Objects (COT) and the Internet of Things (IoT) in our daily life (connected objects, mobile devices, etc.), designing Intelligent Adaptive Distributed software Systems (DIASs) has become an important research issue. Human face the problem of mastering the complexity and sophistication of such systems as those require an important cognitive load for end-users who usually are not expert.

Multiplicity of users, heterogeneity, new usages, decentralization, dynamic execution environments, volumes of information result in new system design requirements: technologies should adapt to users more than users should do to technologies. In the domain of home automation for example, both end-users and system developers face problems:

– end-users: instructions accompanying the devices are too complex and it is hard for non-expert users to master the whole behavior of the system; such systems are usually designed to meet general requirements through a set of predefined configurations (a limited number of scenarios in the best case). A user may need a set of services in a given context and a different set of services in another context. A user does not need to use all what a system could provide in terms of information or services.
– developers lack software support that help them build home automation systems able to adapt to end-users. Self-adaptation mechanisms are not mature yet and most existing support approaches are either too specific or too abstract to be helpful as stated in [1].

All along the paper, we use a simple example of home automation system (see Fig. 1). To illustrate our purposes. Let us consider a system composed of a roller shutter (actuator) and a control key composed of two buttons (sensors).

In the general case and in a manual mode, with a one-button control key, a person can: bring the shutter either to a higher or to a lower position. With a second button, the user can tune inclination of the shutter blades to get more or less light from the outside. As the two buttons cannot be activated at the same time, the user must proceed in two times: first, obtain the desired height (e.g. 70%) then the desired inclination (e.g. 45%). For such systems, three roles are generally defined: *system developer*, *system configurator* and *end-user*. Assume an end-user is at his office and that according to time and weather, his/her requirements for the shutter change (height and inclination). This would solicit

Software System

Fig. 1. An example of home automation system.

the end-user all along the day and even more when there are several shutters with different exposure to the sun. From a developer's point of view, very few support is available to easily construct adaptive systems: when provided, such support is too specific and cannot be easily reused in another context. Adaptation mechanisms and intelligence are generally merged with the application objects which make them difficult and costly to reuse in another application or domain.

Starting from the principle that it is to technology-based systems to adapt to end-users and not the reverse, we address the issue of how to help developers design and produce such systems. We then propose WOF, an object oriented Framework founded on the concept of *Wise Object* (WO), a metaphor to refer to human introspection and learning capabilities.

To make systems able to learn by themselves, we designed introspection, monitoring and analysis software mechanisms such that WOs can learn and construct their own knowledge and experience.

According to this approach (see Fig. 2), "wise" buttons and shutters would gradually construct their experience (e.g. by recording effect of service invocation on their state, statistics on invoked services, etc.) and adapt their behavior according to the context (e.g. physical characteristics of a room, an abstract state defined by a set of data related to the weather, the number of persons in the office, etc.). From the development perspective, we separate in the WOF objects' "wisdom" and intelligence logic (we name *abilities*) from objects' application services (we name *capabilities*) they are intended to render.

To provide application developers with relevant support, we designed WOF with the minimum intrusion in the application source code. Adaptation and distribution related mechanisms defined in WOF can be inherited by application

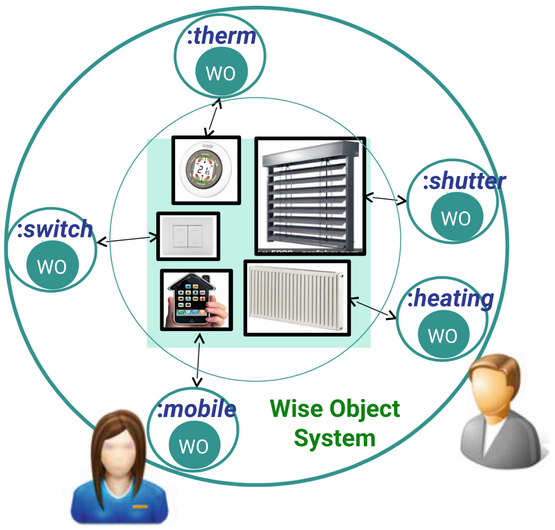


Fig. 2. A wise home automation system.

classes. In our Java implementation of WOF, object classes produced by a developer inherit the behavior of *Wise Object* (WO) class. An instantiated system is a *Wise Object System* (WOS) composed of WOs that interact through an event bus. In the first version of WOF, a WO was able to use introspection and monitoring built-in mechanisms to construct knowledge on: (a) services it is intended to render; (b) the usage done of its services. In the current version, we integrated an event-based WO simulator and a set of Analyzer classes to provide a WO with the possibility to use different analysis models and methods on its data. Our major goal is that a WO can be able to identify common usage of its services and to detect unusual usage. We use the metaphor of emotions to refer to unusual behavior (stress, surprise, etc.). We show in the paper a first experiment based on a statistical analysis method founded on stationary processes to identify usual/unusual behavior.

In the paper, we focus mainly on the architecture of the WO and WOS, their global structure and behavior. In Sect. 2, we discuss the challenges and requirements for DIASs. Then we present design principles and fundamental concepts underlying WOF in Sect. 3. In Sect. 4 we detail the structure and behavior of a WO and a WO System (WOS) and present the architectural patterns we adopted in our design. We focus in section Sect. 5 on WO knowledge analysis using statistical approaches, in particular to identify common usage and detect unusual behavior. To illustrate how to use the WOF, we give an example in the home automation domain in Sect. 6. Finally, in Sect. 7 we discuss our approach and conclude with ongoing work and some perspectives.

2 Requirements

A technology-based system should be able to: (1) *know by itself on itself*, i.e. to learn how it is intended to behaves, to consequently reduce the learning effort needed by end-users (even experimented ones); (2) *know by itself on its usage* to adapt to users' habits. In addition like any service-based system (3) such system should be able to improve the quality of services it is offering. WOF aims at helping developers producing such systems while meeting end-users' requirements:

– *Requirement 1*: We need *non-intrusive* systems that serve users while requiring *just some* (and not all) of their attention and only when necessary. This contributes to *calm-technology* [2] that *describes a state of technological maturity where a user's primary task is not computing, but being human.* As claimed in [3], new technologies might become highly *interruptive* in human's daily life. Though *calm-technology* has been proposed first by Weiser and Brown in early 90's [2], it remains a challenging issue in technology design.
– *Requirement 2*: We need systems composed of *autonomous* entities that are able to independently adapt to a changing context. If we take two temperature sensors installed respectively inside and outside the home, each one reacts differently based on its own experience (knowledge). A difference in temperature that is considered as normal outside (e.g. $5°$) is considered as significant inside. Another situation is when an unexpected behavior occurs, for example a continuous switching on - switching off of a button. In such a case, the system should be able to identify unusual behavior according to its experience and to decide what to do (e.g. raising an alert);
– *Requirement 3*: In an ideal world, an end-user declares his/her needs (a goal) and the system looks for the most optimal way to reach it. This relates to goal-oriented interaction and optimization. The home automaton system user in our example would input the request "I want the shutter at height h and inclination i" and the system based on its experience would choose the "best" way to reach this state for example by planning a set of actions that could be the shortest one or the safest or the less energy consuming, etc. according to the non-functional quality attributes that have been considered while designing the system [4].

Many approaches are proposed to design and develop the kinds of systems we target: multi-agent systems [5], intelligent systems [6], self-X systems [7], adaptive systems [8]. In those approaches, a system entity (or agent) is able to learn on its environment through interactions with other entities. Our aim is to go a step forward by enhancing a system entity with the ability to learn by its own on the way it has been designed to behave. There are at least two advantages to this: (a) as each entity evolves independently from the others, it can control actions to perform at its level according to the current situation. This enables a decentralized control in the system; (b) each entity can improve its performance and then the performance of the whole system, i.e. a collaborative performance.

While valuable, existing design approaches are generally either domain-specific or too abstract to provide effective support to developers. The IBM MAPE-K known cycle for autonomic computing [9] is very helpful to understand required components for self-adaptive systems but still not sufficient to implement them. Recently, more attention has been given to design activities of self-adaptive systems: in [1], authors propose design patterns for self-adaptive systems where roles and interactions of MAPE-K components are explicitly defined. In [10] authors propose a general guide for developers to take decisions when designing self-adaptive systems. Our goal is to offer developers an object oriented concrete architecture support, ready to use for constructing wise systems. We view this as complementary to the work results cited above where more abstract architectures have been defined.

From a system development perspective, our design decisions are mainly guided by the following characteristics: software support should be non-intrusive, reusable and generic enough to be maintainable and used in different application domains with different strategies. Developers should be able to use the framework with the minimum of constraints and intrusion in the source code of the application. We consequently separated in the WOF the objects' "wisdom" and intelligence logic (we name *abilities*) from application services (we name *capabilities*) they are intended to render.

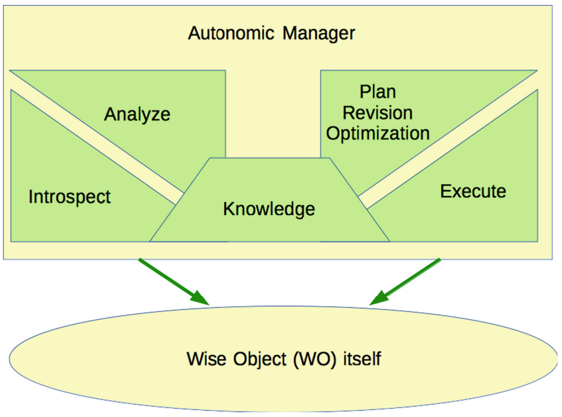
3 Fundamental Concepts of WOF

We introduce the fundamental concepts of WO and WOS from a runtime perspective. We adapt to this end the IBM MAPE-K known cycle for autonomic computing [9].

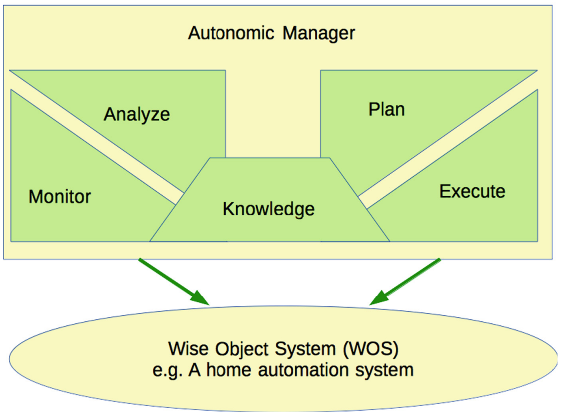
3.1 Concept of WO

We define a Wise Object (WO) as a software object able to learn by itself on itself and on its environment (other WOs, external knowledge), to deliver expected services according to the current state and using its own experience. *Wisdom* refers to the experience such object acquires by its own during its life. We intentionally use terms dedicated to humans as a metaphor. A *Wise Object* is able to learn on itself using introspection. A *Wise Object* is considered as a software avatar intended to "connect" to either a physical entity/device (e.g. a vacuum cleaner) or a logical entity. In the case of a vacuum cleaner, the WO could learn how to clean a room depending on its shape and dimensions. In the course of time, it would in addition improve its performance (less time, less energy consumption, etc.).

- its autonomy: it is able to behave with no human intervention;
- its adaptiveness: it changes its behavior when its environment changes;
- its intelligence: it observes itself and its environment, analyzes them and uses its knowledge to decide how to behave (introspection and monitoring, planning);



(a) WO Dream IAPE-K



(b) WO Awake MAPE-K

Fig. 3. WO MAPE-Ks [11].

– its ability to communicate: with its environment that includes other WOs and end-users in a decentralized way (i.e. different locations).

We designed WO in a way its behavior splits into two states we named *Dream* and *Awake*. The former is dedicated to introspection, learning, knowledge analysis and management when the WO is not busy with service execution. The latter is the state the WO is in when it is delivering a service to an end-user or answering an action request from the environment. The WO then monitors such execution and usage done with application services it is responsible for. We use the word *Dream* as a metaphor for a state where services invoked by the WO do not have any impact on the real system: this functions as if the WO is disconnected from the application device/component/object it is related to.

To ensure adaptiveness, each WO has a set of mechanisms that allow it to perform a kind of MAPE-K loops [9]. *Dream* and *Awake* MAPE-K are respectively depicted by Fig. 3(a) and (b). Let us call the dream MAPE-K a IAPE-K, due to the fact that in the dream case the Monitoring is actually Introspection.

When dreaming, a WO introspects itself to discover services it is responsible for, analyzes impact of their execution on its own state and then plans revision actions on its knowledge. WO constructs its experience gradually, along the dream states. This means that WO knowledge is not necessarily complete and is subject to revisions. Revision may result in adaptation, for instance recording a new behavior, or in optimization like creating a shortening among an action list to reach more quickly a desired state.

When awake, a WO observes and analyzes what and how services are invoked and in what context. According to its experience and to analysis results, a WO is able to communicate an *emotion* if necessary. We define a WO emotion as a distance between the common usage (usual behavior) and the current usage of its services. According to this metaphor, a WO can be surprised if one of its services is used while it has never been the case before. A WO can stress if one of its services is more frequently used or conversely, a WO can be bored. WO emotions are intended to be used as a new information by other WOs and/or the end-users. This is crucial to adaptation at a WOS level (e.g. managing a new behavior) and to attract attention on potential problems (e.g. alerts when temperature is unusually too high). With respect to its emotional state, a WO plans relevant actions (e.g. raising alarms, opening windows and doors, cutting off electricity, etc.).

3.2 Concept of WOS (WO System)

We define a WOS as a distributed object system composed of a set of communicating WOs. Communicated data/information (e.g. emotions) are used by the WOS to adapt to the current context. It is worth noting that each WO is not aware of the existence of other WOs. WOs may be on different locations and it is the charge of the WOS to handle data/information that coordinate WOs' behaviors. The way this is done is itself an open research question. In our case, we defined the concept of *Managers* (see Sect. 4) to carry out communication and coordination among WOs. This is close to the *Implicit Information Sharing Pattern* introduced in [12].

4 Design Models of WO and WOS

WOF is an object oriented framework built on the top of a set of interrelated packages. This section introduces our design model of the concepts presented in the previous section.

4.1 Design Model of WO

Figure 4 shows the UML Class diagram for WO. This model is intentionally simplified and highlights the main classes that compose a WO. WO Class is an abstract class that manages the knowledge of its sub-classes. Knowledge managed by a WO is of two kinds: capability-related (i.e. knowledge on application services) and usage-related (knowledge on service usage). In our present experiment, we have chosen a graph-based representation for knowledge on WO capabilities and usage done of them. Knowledge on WO capabilities is expressed as a state-transition graph while that on WO usage is expressed as a Markov graph where usage-related statistics are maintained. The Markov graph clearly depends on the usage of an object (a WO instance) as 2 WO instances of a same class may be used differently.

Let us recall that WO behavior is split into two states. The dream state and the awake state, see Fig. 5. The dream state is dedicated to acquiring the capability knowledge and to analyzing the usage knowledge. The awake state is the state where the WO executes its methods invoked by other objects or by itself, and, monitors such execution and usage.

To build capability-related knowledge, the WO executes the methods of its sub-class (i.e. the application class) to know their effect on the attributes of this sub-class. This knowledge is itself represented by a state-transition diagram. Each set of attribute values produces a state in the diagram and a method invocation produces a transition. The main constraint in this step is that method invocation must have no real effect on other objects of the application when the WO is dreaming. This is possible thanks to the system architecture described in Sect. 4.2.

Regarding usage-related knowledge on an application object, two kind of situations are studied: emotions and adaptation of behavior.

As introduced in Sect. 3, an emotion of a WO is defined as a distance between its current usage and its common usage. WO can be stressed if one of its methods is more frequently used or conversely, a WO can be bored. WO can be surprised if one of its method is used and this was never happened before.

When a WO expresses an emotion, this information is caught by the WOS and/or other WOs and that may consequently lead to behavior adaptation. At the object level, two instances of the same class that are used differently – different frequencies, different methods... – may have different emotions, thus, different behavior and interaction within the WOS.

A WO uses its capability-related knowledge to compute a path from a current state to a known state [13]. According to the frequency of the paths used, a WO can adapt its behavior. For example, if a path is often used between non-adjacent states, the WO can build a shortcut transition between the initial state and the destination state; it then can also build the corresponding method within its subclass instance (application object). This modifies the capability-related graph of this instance.

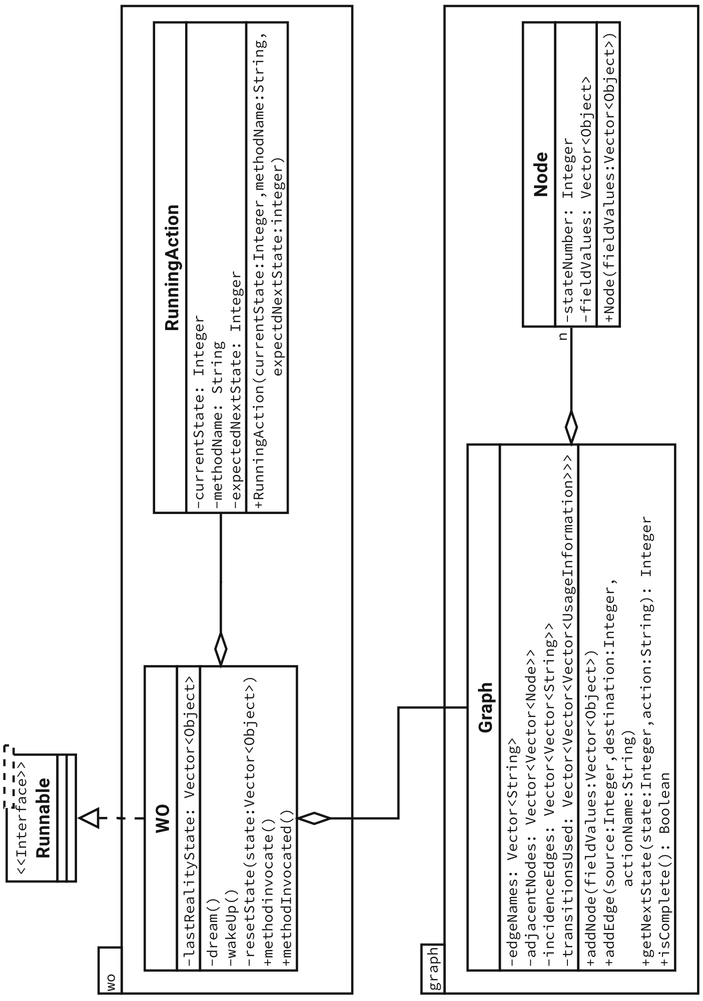
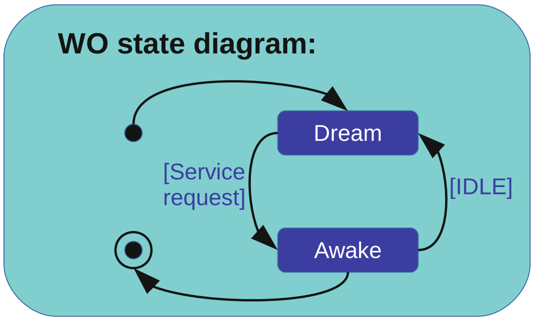
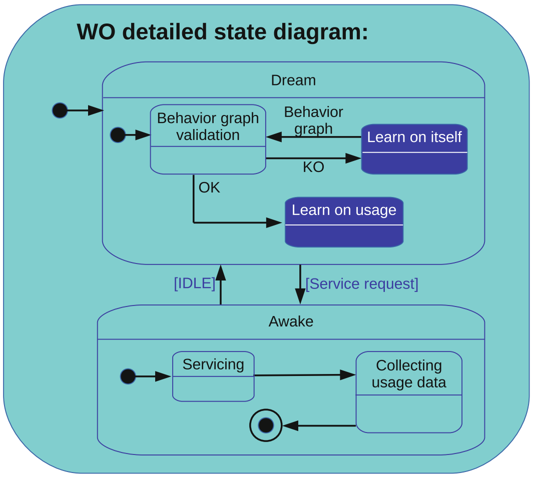

wo
<<Interface>>
Runnable
WO
-lastRealityState: Vector<Object>
-dream()
-wakeUp()
-resetState(state:Vector<Object>)
+methodinvocate()
+methodInvocated()
RunningAction
-currentState: Integer
-methodName: String
-expectedNextState: Integer
+RunningAction(currentState:Integer,methodName:String, expectdNextState:integer)
graph
Graph
-edgeNames: Vector<String>
-adjacentNodes: Vector<Vector<Node>>
-incidenceEdges: Vector<Vector<String>>
-transitionsUsed: Vector<Vector<Vector<UsageInformation>>>
+addNode(fieldValues:Vector<Object>)
+addEdge(source:Integer,destination:Integer, actionName:String)
+getNextState(state:Integer,action:String): Integer
+isComplete(): Boolean
Node
-stateNumber: Integer
-fieldValues: Vector<Object>
+Node(fieldValues:Vector<Object>)
n

Fig. 4. UML class diagram of WO.



(a) WO short state diagram



(b) WO detailed state diagram

Fig. 5. UML state diagram of WO built-in behavior [11].

4.2 Design Model of WOS

As explained in Sect. 3, WOs are not aware of the existence of other WOs. They are distributed and communicate data/information towards their environment. WOs may be on different locations and one or many *Managers* carry out communication and coordination among them. In this paper, we propose a concrete architecture based on a bus system, where any WO communicates with other objects through the bus. This architecture has many advantages.

A first one is the scalability. It is easier to add WOs, managers, loggers... on this kind of architecture than to modify a hierarchical architecture. Moreover, this architecture is obviously distributed and enables distribution/decentralizion of WOs in the environment.

The third main advantage is the ability for a WO to disconnect/reconnect from/to the bus when needed. This makes it possible the implementation of the Dream state (see Sect. 3). Let us recall that in the dream state, a WO can invoke its own methods to build its capability-related graph, but these invocations must not have any effect on the subject system.

Thus, when a WO enters the Dream state, it disconnects itself from the bus and can invoke its methods without impact on the real world system. More precisely, the WO disconnects its "sending system" from the bus, but it continues receiving data/information via the bus. Therefore, if a WO in Dream state receives a request, it reconnects to the bus, goes out from the Dream state to enter into Awake state and serves the request.

Figure 6 shows the UML class diagram of a bus-based WO system. This model is simplified and highlights the main classes. The system uses an Event/Action mechanism for WOs' interactions. On an event, a state change occurs in a WO, an action may be triggered on another WO. The peers "Event/Action" are defined by Event, Condition, Action (ECA) rules that are managed by a Manager. When this latter catches events (StateChangeEvent), it checks the rules and conditions and posts a request for action (ActionEvent) on the bus. From the WO point of view, if one of its subclass instance state changes at Awake state, it posts a StateChangeEvent on the bus. When a WO receives an ActionEvent, two cases may occur: either the WO is in Awake state or in Dream state. If the WO is in Awake state, it goes to the end of its current action and starts the action corresponding to the received request. If the WO is in Dream state, it stops dreaming and enters into the Awake state to start the action corresponding to the received request.

In our Java implementation of WOF, object classes produced by a developer inherit the behavior of *Wise Object* (WO) class. An instantiated system is defined as a *wise system* composed of *Wise Objects* that interact through a (or a set of distributed) *Manager(s)* implemented by an event-based bus according to *publish-subscribe* design pattern.

4.3 Design Model of WO Data Analyzers

As stated all along the paper, a WO is able to collect and analyze usage-related data. To enrich the WOF framework and to offer the possibility to use different analysis models and methods on the same data, we associate a WO with a set of Analyzer classes according to a Factory-like design pattern [14]. This design decision aims at defining a set of analyzers on collections of data (i.e. instances of "Graph") with the possibility to compare analysis results.

Therefore an Analyzer class named "daClass" is statically registered into the WO class using "registerDAClass(daClass: Class<DataAnalyzer>)" static method, where "DataAnalyzer" is the abstract class for analyzers. Analyzers of a WO – the analyzers of knowledge graphs of the WO – are then instantiated in the WO constructor. Figure 7 illustrates this design model. This approach is close to factory pattern where the factory is the WO class, the "DataAnalyzers" are the products and the WO instances are the clients.

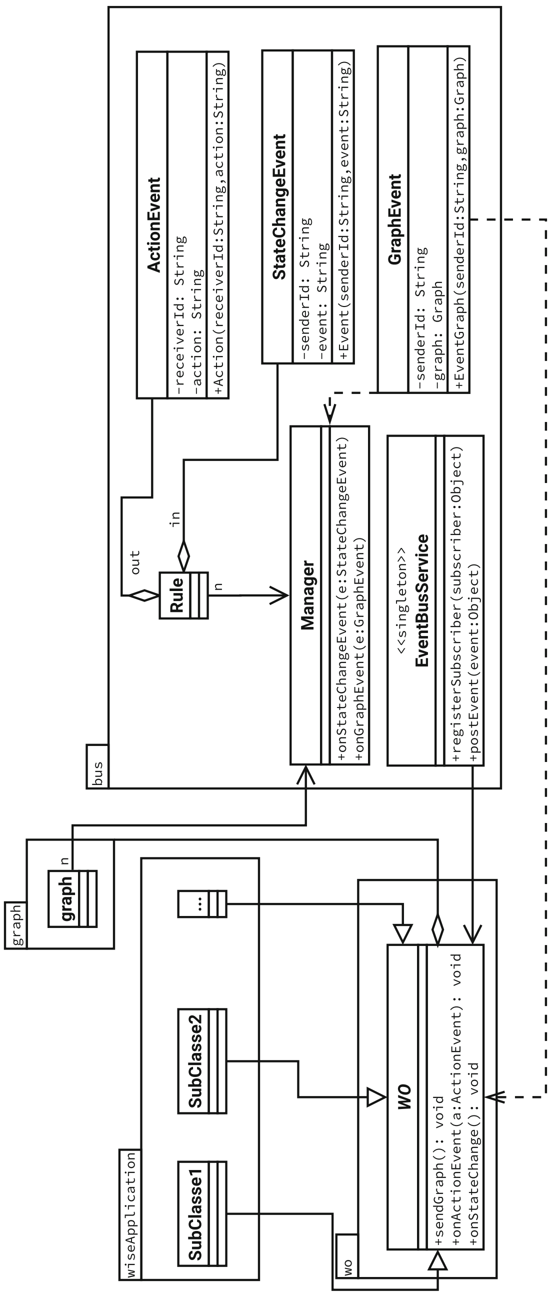


Fig. 6. UML class diagram of a bus-based WO system.

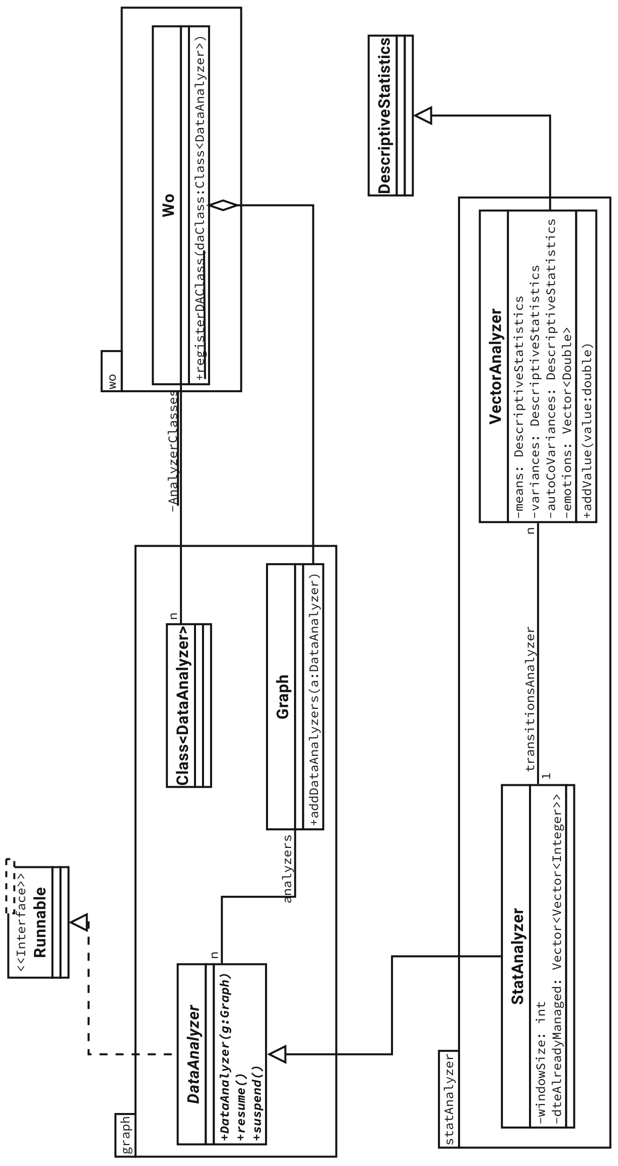


Fig. 7. UML class diagram of WO data analyzer pattern.

The “DataAnalyzer” abstract class implements the “Runnable” interface so that an analyzer is implemented by an independent thread. This abstract class defines 3 abstract methods: “DataAnalyzer(g:Graph)”, “resume()” and “suspend()”. The first “DataAnalyzer(g:Graph)” is the default constructor that requires the graph of knowledge to analyze. The second “resume()” starts or resumes the analyzer. Let us recall that the analysis – the learning activity – only occurs within the dream state of a WO. Therefore, the analyzers must be stopped and resumed accordingly. The last method “suspend()” suspends the analyzer. As depicted by Fig. 7, we realized an implementation of the “DataAnalyzer” abstract class: “StatAnalyzer”.

The “StatAnalyzer” performs a statistical analysis of events: occurrences of graph transitions. It stores for each transition the dates of its occurrences in a “VectorAnalyzer”. Each “VectorAnalyzer” of a “StatAnalyzer” is characterized by a “windowSize” that represents the memory size of the vector. When the vector is full, if a new event occurs, the oldest is forgotten: removed from the vector. Moreover, as the analyzer only performs the analysis during the dream states of a WO, it also stores information about the data that has already been analyzed to resume analysis from where it stopped at the last suspend. Regarding the “VectorAnalyzer”, this class extends the descriptive statistics library of the “The Apache Commons Mathematics Library”. The descriptive statistics provides a dataset of values of a single variable and computes descriptive statistics based on stored data. The number of values that can be stored in the dataset may be limited or not. The “VectorAnalyzer” extends this class to provide statistics about the evolution of means, variances and autocorrelations, when a new value is added into the dataset. A representation of this evolution – a distance with the stationarity – is stored in the “emotion” vector. The next section describes this analysis approach.

5 Statistical Analysis and WO Emotions

The first analyzer we implements is based on the weaker forms of stationarity of the process [15]. The stationarity study focuses on the occurrences of the event. We call event the invocation of a method from a given state (i.e. the execution of a transition of the knowledge graph). Figure 8 gives a graph of knowledge, with 4 possible events: $t1$, $t2$, $t3$ and $t4$. This events occur at different times, for example, on Fig. 8 the event $t1$ occurs at times $\left[e_{t1}^1, e_{t1}^2, e_{t1}^3 \ldots\right]$ In this first analysis, we do not study the correlation between the events.

Let $x(i)$ a continuous and stationary time random process. The weaker forms of stationarity (WSS) defines that the mean $E\left[x(i)\right]$ and variance $Var\left[x(i)\right]$ do not vary with respect to time and the autocovariance $Cov\left[x(i), x(i-k)\right]$ only depends on range k.

This process is a WSS process if and only if:

$$\begin{array}{l} E\left[x(i)\right] = \mu \quad \forall i, \\ Var\left[x(i)\right] = \sigma^2 \neq \infty \quad \forall i, \\ Cov\left[x(i), x(i-k)\right] = f(k) = \rho_k \quad \forall i \forall k. \end{array}$$

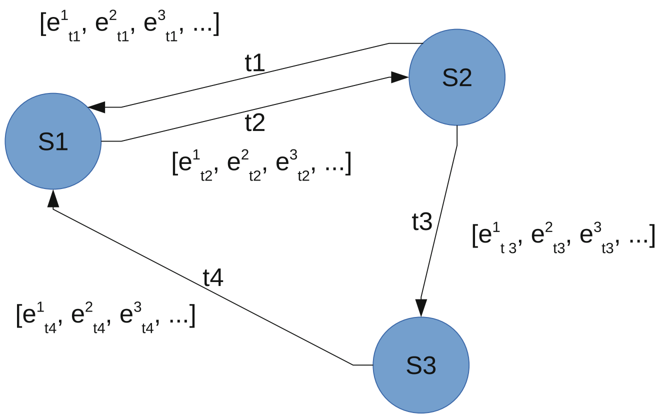


Fig. 8. Example of time series for knowledge graph analysis.

This definition implies the analysis of the whole time series. In our case, the common usage can change and we define it by the stationarity. Therefore, we compute the stationarity – the common usage – on a sliding window of size w:

$$\begin{aligned}&E\left[x(i)\right] = \mu \quad \forall i \in [t-w, t],\\ &Var\left[x(i)\right] = \sigma^2 \neq \infty \quad \forall i \in [t-w, t],\\ &Cov\left[x(i), x(i-k)\right] = f(k) = \rho_k \quad \forall i \in [t-w, t]\forall k,\end{aligned}$$

where the time series $x(i)$ are the occurrences $\left[e_\tau^{t-w} \dots e_\tau^i \dots e_\tau^t\right]$ of a given event – i.e. transition – τ between t – w and t.

According to this definition of the stationarity, we define an emotion as the distance between the current usage and the common usage, in other words the distance with the stationarity measure. We define this distance $d(x(i))$ by the following centered normalized scale where:

$$d(x(i)) = \begin{cases} d(E\left[x(i)\right]) = \frac{E[x(i)] - \overline{E[x(j)]}}{(\max(E[x(j)]) - \min(E[x(j)]))/2}, \\ d(Var\left[x(i)\right]) = \frac{Var[x(i)] - \overline{Var[x(j)]}}{(\max(Var[x(j)]) - \min(Var[x(j)]))/2}, \\ d(Cov\left[x(i), x(i-k)\right]) = \frac{Cov[x(i), x(i-k)] - \overline{Cov[x(j), x(j-k)]}}{(\max(Cov[x(j), x(j-k)]) - \min(Cov[x(j), x(j-k)]))/2}, \end{cases}$$

where $j \in [t-w, t]$ and $\overline{E\left[x(j)\right]}$, $\overline{Var\left[x(j)\right]}$ and $\overline{Cov\left[x(j), x(j-k)\right]}$ are respectively the means of means, variances and autocovariances on the range $[t-w, t]$.

Thus, when a new event occurs at $t+1$, we compute the distance with the common usage between $t-w$ and t. If all values of the distance – $d(E\left[x(i)\right])$, $d(Var\left[x(i)\right])$ and $d(Cov\left[x(i), x(i-k)\right])$ – are in $[-1, 1]$ this is considered as a common behavior, otherwise this is identified as a behavior change (unusual usage) relatively to the knowledge on the common usage.

6 An Illustrating Example "Home Automation"

The concept of WO has many scopes of application. It can be used to adapt an application to its environment, to monitor an application from inside, to manage an application according to the usage done of it... In this section, we highlight the WO behavior within a home automation application. This choice is justified by the fact that:

– home automation systems are usually based on a bus where many devices are plugged on;
– home automation devices have behavior that can be represented by a simple state diagram.

According to the first point, a home automation system can be directly mapped onto a WO system based on a bus where the home automation devices are related to WOs. The second point avoids the combinatorial explosion that can appear due to a large number of states to manage in a state diagram.

Let us take a simple example of a switch and a shutter. The switch is modeled by 2 states "on" and "off" and 3 transitions "on()" , "off()" and "switch()".

Listing 1.1. Switch Java code.

```
public class Switch extends Wo  {
  public boolean position;

  public Switch() {
    super();
  }
  public void on() {
    invoke();
    position = true;
    invoked();
  }
  public  void off() {
    invoke();
    position = false;
    invoked();
  }
  public  void switch() {
    invoke();
    if(position){
      position = false;
    }else{
      position = true;
    }
    invoked();
  }

}
```

The shutter is modeled by n states that represent its elevation between 0% and 100%. If the elevation is 0%, the shutter is totally closed and if the elevation is 100%, the shutter is totally open. To avoid a continuous system, the shutter can only go up or down step by step.

Listing 1.2. Shutter Java code.

```
public class RollingShutter extends Wo {
  private int elevation = 0;
  private static int step = 20;

  public RollingShutter() {
    super();
  }

  public void down(){
    methodInvocate();
    if(this.elevation >0){
      this.elevation -= RollingShutter.step;
    }
    if(this.elevation <= 0){
      this.elevation = 0;
    }
    methodInvocated();
  }

  public void up() {
    methodInvocate();
    if(!this.elevation < 100){
      this.elevation += RollingShutter.step;
      if (this.elevation >= 100){
    this.elevation = 100;
      }
    }
    methodInvocated();
  }
```

As one design principle behind WOF is to minimize intrusion within the application source code, we have succeeded to limit them to the number of two "warts". The examples highlight those 2 intrusions in the code. They are concretized by two methods implemented in the WO Class – methodInvocate() and methodInvocated() – and must be called at the beginning and the end of any method of the WO subclass (application class). Those methods monitor the execution of a method on a WO instance. We discuss about these "warts" in the last section.

In our example, an instance of Switch and another of RollingShutter are created. Two ECA rules are defined to connect those WOs:

– [Switch*Instance*.on?/True/RollingShutter*Instance*.up()]
– [Switch*Instance*.off?/True/RollingShutter*Instance*.down()]

They define that when the event "on" occurs on the switch, the action – method – "up" must be executed on the rolling shutter and that when the event "off" occurs on the switch, the action "down" must be executed on the rolling shutter. For the experiment and feasibility study, the action on the Switch*Instance* – "on()" and "off()" invocations – are simulated using the WO simulator we are

Switch*Instance*		RollingShutter*Instance*	
on	off	up	down
0	1769	3	1774
1	6015	5	6016
1	6624	5	6625
4263	10435	4264	10436
8523	10435	8525	10444
9963	11026	9968	11028
9964	11026	9966	11028
10994	12615	10997	12616
10995	15811	10996	15816
...	...	...	...

Log 1. First event log stored on each WO.

developing. The actions "on" and "off" occur according to a Poisson distribution and depend on the elevation of the rolling shutter. The likelihood of action "off" occurrence is *RollingShutterInstance.elevation*/100, the likelihood of action "on" occurrence is inversely proportional. When an action occurs, "on" or "off", it can occur x times successively without delay, where x is bounded by the number of occurrences to reach the bound of shutter elevation, respectively 100% and 0%.

Presently, a WO acquires knowledge about its capabilities using a graph representation. The knowledge about its usage is the logs of all its actions/events and can be presented by a Markov graph. The logging presented in Log 1 shows the events occurred on each WO of the system. This information is collected from each WO. With this information each WO can determine its current behavior and a manager can determine the system behavior. This is discussed in Sect. 7. Log 2 gives Markov graph logging representation. Let us note that the Markov graph representation hides time-related information as it is based on the frequency of occurrences. Log 2 shows that the wise part of the Switch instance detects the 2 states and the 6 transitions. It also shows a 2 × 2 adjacency matrix followed by a description of the 6 transitions including their usage-related statistics.

Log 2 shows for instance that from the state 0 with the position attribute at false, the Switch*Instance* may execute method "on()" or "switch()" and go to state 1 or execute method "off()" and remain in the same state 0. Usage-related statistics show that method "switch()" is never used from the state 0 all along the 1000 iterations.

Regarding the RollingShutter instance, the logging after the 2nd iteration (Log 3) and the last Log 4 are given. Log 3 shows that the wise part of the RollingShutter instance detects 6 states and 10 transitions (green values of adjacency matrix). Consequently, it has not detected all the possible transitions yet. This incomplete knowledge is not a problem, during the next Dream state or if it uses those transitions during the Awake state, the WO part of the application object will update its knowledge. The last Log 4 shows that all states and transitions are detected (learnt).

```
Graph:
2 States, 6 Transitions
   0 1
0: 1 1
1: 1 1
State [0 , false] :
      Adjacency on->[1 , true] - 0.313,
                switch->[1 , true] - 0.0,
                off->[0 , false] - 0.687,
State [1 , true] :
      Adjacency off->[0 , false] - 0.311,
                on->[1 , true] - 0.689,
                switch->[0 , false] - 0.0,
Current State: 1
```

Log 2. Switch log after 1000 iterations.

```
Graph:
6 States, 10 Transitions
   0 1 2 3 4 5
0: 1 1 0 0 0 0
1: 1 0 1 0 0 0
2: 0 1 0 1 0 0
3: 0 0 1 0 1 0
4: 0 0 0 1 0 1
5: 0 0 0 0 0 0
State [0 , 0 , 20] :
      Adjacency down->[0 , 0 , 20] - 0.0,
                up->[1 , 20 , 20] - 1.0,
State [1 , 20 , 20] :
      Adjacency up->[2 , 40 , 20] - 1.0,
                down->[0 , 0 , 20] - 0.0,
State [2 , 40 , 20] :
      Adjacency down->[1 , 20 , 20] - 1.0,
                up->[3 , 60 , 20] - 0.0,
State [3 , 60 , 20] :
      Adjacency up->[4 , 80 , 20] - 0.0,
                down->[2 , 40 , 20] - 0.0,
State [4 , 80 , 20] :
      Adjacency down->[3 , 60 , 20] - 0.0,
                up->[5 , 100 , 20] - 0.0,
State [5 , 100 , 20] :
      Adjacency ,
Current State: 1
```

Log 3. Rolling shutter log after the 2nd iteration.

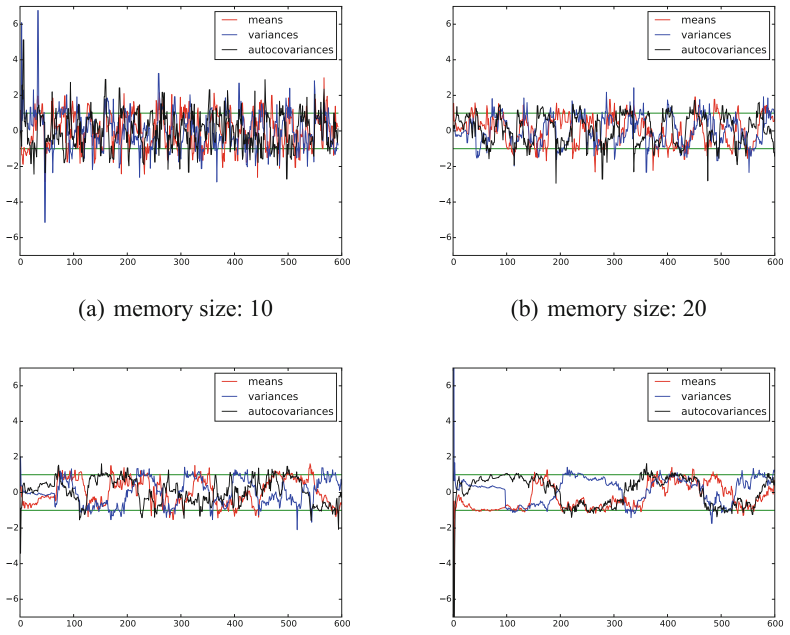
Another analysis of the log is given by Fig. 9. This figure presents the "emotion" for the event "on" from state 0, computed from the statistical analysis presented in Sect. 5. Figures 9(a)–(d) present the analysis of the common usage

```
Graph:
6 States, 12 Transitions
   0 1 2 3 4 5
0: 1 1 0 0 0 0
1: 1 0 1 0 0 0
2: 0 1 0 1 0 0
3: 0 0 1 0 1 0
4: 0 0 0 1 0 1
5: 0 0 0 0 1 1
State [0 , 0 , 20] :
      Adjacency down->[0 , 0 , 20] - 0.0,
                up->[1 , 20 , 20] - 1.0,
State [1 , 20 , 20] :
      Adjacency up->[2 , 40 , 20] - 0.653,
                down->[0 , 0 , 20] - 0.347,
State [2 , 40 , 20] :
      Adjacency down->[1 , 20 , 20] - 0.456,
                up->[3 , 60 , 20] - 0.544,
State [3 , 60 , 20] :
      Adjacency up->[4 , 80 , 20] - 0.443,
                down->[2 , 40 , 20] - 0.557,
State [4 , 80 , 20] :
      Adjacency up->[5 , 100 , 20] - 0.375,
                down->[3 , 60 , 20] - 0.625,
State [5 , 100 , 20] :
      Adjacency up->[5 , 100 , 20] - 0.0,
                down->[4 , 80 , 20] - 1.0,
Current State: 1
```

Log 4. Rolling shutter log after the last iteration.

with different sizes of memory (window size of "VectorAnalyzer"). Between -1 and 1, the behavior is considered as usual, because it already has appeared in the past stored in the memory. Out of the range $[-1..1]$ an unusual behavior is detected relatively to the knowledge in memory and the more important the distance with the range is, the more important the emotion is. Those results are consistent from the data analysis point of view: the bigger the time window is, the smoother the result is. From the emotion point of view, it means that the wider the memory is, the less unusual behavior is detected.

It is worth noticing that this example is intentionally simple as our goal is to highlight the kind of knowledge a WO can currently acquire and analyze. Capability graphs and usage logging are the knowledge base for WOs. We discus the management and use of this knowledge in Sect. 7.



(a) memory size: 10

(b) memory size: 20

(c) memory size: 40

(d) memory size: 80

Fig. 9. Evolution of emotions according to memory size.

7 Discussion and Concluding Remarks

Our work addresses the issue of designing distributed adaptive software systems through WOF: a software object-based framework implemented in Java. At a conceptual level, WOF is built around the concept of "wise object" (WO), i.e. a software object able to: (a) learn on its capabilities (services), (b) learn on the way is being used and (c) perform data analysis to identify common usage and detect unusual behavior. At a concrete level, a WO uses: (a) introspection and monitoring mechanisms to construct its knowledge, (b) an event-based bus to communicate with the system and (c) a set of analyzers to identify both usual and unusual behaviors.

Regarding data analysis, we implemented a first statistical analyzer, based on the theory of stationary processes. This experiment is a step forward towards behavior analysis and emotion representation in adaptive systems. As shown in the paper, an emotion is defined as a distance between an unusual behavior and a common behavior.

Our work and experiments around WOF raised many research issues and perspectives. A first main perspective is to use other knowledge aggregation theories/techniques to represent *emotions* of a *Wise Object*: solutions may involve techniques from information fusion, multi-criterion scales or fuzzy modeling. A second one is to generalize behavior analysis and emotion representation to a WOS (Wise Object System): this requires knowledge aggregation to extract relevant information on the whole system starting from individual WOs.

In the present version of WOF, intrusiveness in application source code is limited to the inheritance relationship and two warts: the WO methods methodInvocate() and methodInvocated() that must be called at the beginning and the end of an application method. Regarding this issue, we envisage different solutions in the next version of WOF: (a) add dynamic Java code on-the-fly at runtime; (b) use Aspect Oriented Programming [16]; (c) use dynamic proxy classes.

We are convinced that *wise systems* are a promising approach to help humans integrate new technologies both in their daily life as end-users and in development processes as system developers. We use home automation to illustrate our work results but those can also apply to other domains like health that heavily rely on human expertise. Authors in [17,18] propose interactive Machine Learning (iML) to solve computationally hard problems. With regard to this, WOF puts the "human-in-the-loop" in two cases: when defining ECA rules to connect distributed WOs and when validating capability-related knowledge constructed by WOs.

To validate our approach, we recently initiated a new internal project called COMDA whose aim is to study and test our research ideas on a real system. The latter consists of a set of connected objects (sofa, chairs, etc.) to identify unusual situations like "no life sign in the living-room this morning" which is crucial in the domain of person ageing in place.

References

1. Abuseta, Y., Swesi, K.: Design patterns for self adaptive systems engineering. CoRR abs/1508.01330 (2015)
2. Weiser, M., Brown, J.S.: Designing calm technology. PowerGrid J. **1**(01) (1996)
3. Amber Case: Amber case 2011, we are all cyborgs now. TED Talk (2010)
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, Reading (c1998). Online version: Bass, L.: Software Architecture in Practice. Addison-Wesley, Reading c1998 (OCoLC)605442178
5. Wooldridge, M.: An Introduction to MultiAgent Systems, 2nd edn. Wiley Publishing, Hoboken (2009)
6. Roventa, E., Spircu, T.: Management of Knowledge Imperfection in Building Intelligent Systems. Studies in Fuzziness and Soft Computing. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77463-1
7. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing—degrees, models, and applications. ACM Comput. Surv. **40**, 7:1–7:28 (2008)
8. Salehie, M., Tahvildari, L.: Self-adaptive software: landscape and research challenges. ACM Trans. Auton. Adapt. Syst. **4**, 14:1–14:42 (2009)
9. IBM: An architectural blueprint for autonomic computing. Technical report, IBM (2005)
10. Brun, Y., Desmarais, R., Geihs, K., Litoiu, M., Lopes, A., Shaw, M., Smit, M.: A design space for self-adaptive systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-adaptive Systems II. LNCS, vol. 7475, pp. 33–50. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_2

11. Alloui, I., Vernier, F.: A wise object framework for distributed intelligent adaptive systems. In: ICSOFT 2017, Madrid, Spain (2017)
12. Weyns, D., et al.: On patterns for decentralized control in self-adaptive systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-adaptive Systems II. LNCS, vol. 7475, pp. 76–107. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_4
13. Moreaux, P., Sartor, F., Vernier, F.: An effective approach for home services management. In: 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Garching, pp. 47–51. IEEE (2012)
14. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Longman Publishing Co. Inc., Boston (1995)
15. Lindgren, G.: Stationary Stochastic Processes: Theory and Applications. Texts in Statistical Science. Chapman and Hall, New York (2012)
16. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: Akşit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0053381
17. Holzinger, A.: Interactive machine learning for health informatics: when do we need the human-in-the-loop? Brain Inform. **3**, 119–131 (2016)
18. Holzinger, A., Plass, M., Holzinger, K., Crişan, G.C., Pintea, C.-M., Palade, V.: Towards interactive machine learning (iML): applying ant colony algorithms to solve the traveling salesman problem with the human-in-the-loop approach. In: Buccafurri, F., Holzinger, A., Kieseberg, P., Tjoa, A.M., Weippl, E. (eds.) CD-ARES 2016. LNCS, vol. 9817, pp. 81–95. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45507-5_6

Check for updates

Classifying Big Data Analytic Approaches: A Generic Architecture

Yudith Cardinale[1](✉), Sonia Guehis[2,3], and Marta Rukoz[2,3]

[1] Dpto. de Computación y TI, Universidad Simón Bolívar, Caracas 1080-A, Venezuela
ycardinale@usb.ve
[2] Université Paris Nanterre, 92001 Nanterre, France
[3] Université Paris Dauphine, PSL Research University, CNRS, UMR[7243], LAMSADE, 75016 Paris, France
{sonia.guehis,marta.rukoz}@dauphine.fr

Abstract. The explosion of the huge amount of generated data to be analyzed by several applications, imposes the trend of the moment, *the Big Data boom*, which in turn causes the existence of a vast landscape of architectural solutions. Non expert users who have to decide which analytical solutions are the most appropriates for their particular constraints and specific requirements in a Big Data context, are today lost, faced with a panoply of disparate and diverse solutions. To support users in this hard selection task, in a previous work, we proposed a *generic architecture* to classify Big Data Analytical Approaches and a *set of criteria* of comparison/evaluation. In this paper, we extend our classification architecture to consider more types of Big Data analytic tools and approaches and improve the list of criteria to evaluate them. We classify different existing Big Data analytics solutions according to our proposed generic architecture and qualitatively evaluate them in terms of the criteria of comparison. Additionally, we propose a preliminary design of a decision support system, intended to generate suggestions to users based on such classification and on a qualitative evaluation in terms of previous users experiences, users requirements, nature of the analysis they need, and the set of evaluation criteria.

Keywords: Big Data Analytic · Analytic models for big data
Analytical data management applications

1 Introduction

The Big Data phenomenon revolutionized and impacted the modern computing industry, which have reviewed their policies, architectures, and their production environment to support a continuous increase on the computational power that produces an overwhelming flow of data [1]. Big Data databases have recently become important NoSQL (being non-relational, distributed, open-source, and horizontally scalable) and NewSQL (taking the advantages of relational and

© Springer International Publishing AG, part of Springer Nature 2018
E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 268–295, 2018.
https://doi.org/10.1007/978-3-319-93641-3_13

NoSQL systems) data repositories in enterprises as the center for data analytics, while enterprise data warehouses (EDWs) continue to support critical business analytics [2,3]. This scenario induced a paradigm shift in the large scale data processing and data mining, computing architecture, and data analysis mechanisms. This new paradigm has spurred the development of novel solutions from both industry (e.g., analysis of web-data, clickstream, network-monitoring logs) and science (e.g., analysis of data produced by massive-scale simulations, sensor deployments, telescopes, particle accelerators, genome sequencers) [4,5].

In this context, analytical data management applications, affected by the explosion of the amount of generated data, are shifting away their analytical databases towards a vast landscape of architectural solutions combining storage techniques, programming models, languages, and tools. In this scenario, non expert users who have to decide which analytical solution is the most appropriate for their particular constraints and specific requirements in a Big Data context, is today lost, faced with a panoply of disparate and diverse solutions.

To support users in this hard selection task, in a previous work [6], we proposed: (i) a *generic architecture to classify Big Data Analytical Approaches*, focused on the data storage layer, the parallel programming model, the type of database, and the query language that can be used; these aspects represent the main features that allow to distinguish classical EDWs from analytical Big Data approaches; and (ii) a *set of criteria of comparison/evaluation*, such as On-Line Analytical Processing (OLAP) support, scalability, and fault tolerance support. In this paper, we extend our previous proposed classification architecture to consider a more wide typology of Big Data tools and approaches and we improve the list of evaluation criteria, by including other important aspects for analytic processing, such as Machine Learning support. We classify different existing Big Data analytics solutions according to our proposed generic architecture and qualitatively evaluate them in terms of the criteria of comparison. Additionally, in this work we propose a preliminary design of a decision support system. The decision support system is intended to generate suggestions to users based on such classification and on a qualitative evaluation in terms of previous users experiences, users requirements, nature of the analysis they need, and the set of evaluation criteria. This work represents another step towards the construction of a more sophisticate Decision Support System that will help non-expert users in selecting an appropriate Big Data Analytic Approach.

Our contributions are presented as follows. Section 2 presents a review of most popular programming models, with their respective processing frameworks, and query languages Big Data analytics. Our new extended architecture and classification for analytic approaches are presented in Sect. 3. In Sect. 4, diverse solutions for analytic Big Data are studied. Section 5 presents the related work. Section 6 presents the preliminary design of a decision support system. We finally conclude in Sect. 7.

2 Preliminaries

The large-scale parallel Big Data processing scenario has brought new challenges in the programming models in order to process and analyze such huge amount of data. It is necessary a new model of cluster computing, in which data-parallel computations can be executed on clusters of unreliable machines. Currently, there exist popular processing frameworks that materialize a specific programming model and automatically provide locality-aware scheduling, fault tolerance, and load balancing. While programming models and processing frameworks are most focused on supporting the implementation of complex parallel algorithms and on the efficient distribution of tasks, there exists another axis related to querying and analyzing that huge amount of data. Several languages have been proposed with the intention of improving the programming efficiency for task description and dataset analysis. In this section we recall the most popular and recent programming models and frameworks for Big Data processing and the classification of query language that we proposed in our previous work [6].

2.1 Big Data Programming Models

Classical parallel programming models, such as master-slave with Message Passing Interface (MPI) and multithreading with Open Multi-Processing (OpenMP), are not adequate for Big Data scenarios due to the high network bandwidth demanded to move data to processing nodes and the need to manually deal with fault tolerance and load balancing [4]. However, inspired on them, there have been deployments of cluster computing models, which aggregate computational power, main memory, and I/O bandwidth of shared-nothing commodity machines, combined with new parallel programming models [7,8]. In general, the underlying system that implements the programming model (i.e., the processing framework), also manages the automatic scheduling, load balancing, and fault tolerance without user intervention. The main difference between classical parallel models and the new ones is that, instead of moving data, the processing functions are taken to the data.

- **The Pioneer: MapReduce Model.** The most popular model of data parallel programming in the context of Big Data is MapReduce [9]. Along with it, Hadoop[1] is its most popular core framework implementation for carrying out analytic on Big Data. With simple distributed functions (based on Lisp primitives), the idea of this parallel programming model is to combine `map` and `reduce` operations with an associated implementation given by users (i.e., user-defined functions – UDF) and executed on a set of `n` nodes, each with data. Hadoop framework is in charge of splitting input data into small chunks, designated as key/value pairs, storing them on different compute nodes, and invoking `map` tasks to execute the UDF. The `map` tasks generate intermediate key/value pairs according to the UDF. Subsequently, the framework initiates

[1] http://hadoop.apache.org.

a Sort and Shuffle phase to combine all the intermediate values related to the same key and channelizes data to parallel executing `reduce` tasks for aggregation. MapReduce has inspired other models that extend it to cater different and specific needs of applications, as we explain in the following.

- **MapReduce with Parallelization Contracts: PACT Model.** In [10], it is proposed a programming model which extends the MapReduce model with Parallelization Contracts (PACT) and additional second-order functions. PACT programs are represented by Directed Acyclic Graphs (DAGs), in which edges represent communication channels that transfer data between different subprograms and vertices are sequential executable programs that process the data that they get from input channels and write it to output channels. Nephele/PACT [11] is a distributed execution framework that supports PACT programs execution. A PACT consists of one UDF which is called Input Contract and an optional Output Contract. UDFs for Input Contract are similar to `map` and `reduce`, but additionally user can also define other type of Input Contracts, such as `cross`, `CoGroup`, and `match`. Output Contracts are optional and can be used for optimizations. The main differences between MapReduce and PACT programming models are: (i) PACT allows additional functions that fit more complex data processing tasks, which are not naturally and efficiently expressible as map or reduce functions, as they occur in fields like relational query processing or data mining; (ii) MapReduce systems tie the programming model and the execution model (conducting sub-optimal solutions); with PACT, it is possible to generate several parallelization strategies, hence offering optimization opportunities; and (iii) MapReduce loses all semantic information from the application, except the information that a function is either a map or a reduce, PACT model preserves more semantic information through a larger set of functions.
- **MapReduce with "in memory" Operations: Models for real time analytics.** To ensure real time processing, new programming models propose "in-memory" operations. The Spark system [12] was the first proposal of this kind of programming model, more recently the Apache Storm project[2] and the Apache Flink project[3] propose other "in-memory" programming models. The Spark programming model, unlike acyclic data flows problems (such those treated with MapReduce and PACT), focuses on applications that reuse a working set of data across multiple parallel operations. It provides two main abstractions for parallel programming: Resilient Distributed Datasets (RDDs) and parallel operations on these datasets (invoked by passing a function to apply on a dataset). The RDD is a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Therefore, users can explicitly cache an RDD in memory across machines and reuse it in multiple MapReduce-like parallel operations. Storm, as PACT, is based on DAG that represents a topology, composed by spouts, bolts, and streams. In a topology, spouts and bolts act as the vertices of the graph, a spout acts

[2] http://storm.apache.org.
[3] https://flink.apache.org.

as a data receiver from external sources and creator of streams for bolts to support the actual processing. Streams are defined as unbounded data that continuously arrives at the system and are represented as the edges of the DAG. Thus, a topology acts as a pipeline to transform data. Similar to Storm, the Flink framework is an hybrid engine that effectively supports both batch and real-time processing, but focused on streaming first. The components of the stream processing model in Flink include streams, operators, sources, and sinks. Flink manages MemorySegments, a distribution unit of memory (represented by a regular Java byte array). Beside programming model which allow iterative algorithms (i.e., cyclic data flows), Spark, Storm, and Flink overcome MapReduce and PACT by handling most of their operations "in memory".

2.2 A Classification of Big Data Query Languages

The wide diversification of data store interfaces has led the loss of a common programming paradigm for querying multistore and heterogeneous repositories and has created the need for a new generation of special federation between Hadoop-like Big Data platforms, NoSQL and NewSQL data stores, and EDWs. Mostly, the idea of all query languages is to execute different queries to multiple, heterogeneous data stores through a single query engine. We have classified these query languages in Procedural Languages, Language Extensions, and SQL-like Declarative Languages [6,13].

- **Procedural languages.** This type of query languages are built on top of frameworks that do not provide transparent functions (e.g., `map` and `reduce`) and cannot cover all the common operations. Therefore, programmers have to spend time on programming the basic functions, which are typically hard to maintain and reuse. Hence, simpler procedural language have been proposed. The most popular of this kind of languages are Sawzall of Google [14], PIG Latin of Yahoo! [15], and more recently Jaql [16]. They are domain-specific, statically-typed, and script-like programming language used on top of MapReduce. These procedural languages have limited optimizations built in and are more suitable for reasonably experienced analysts, who are comfortable with a procedural programming style, but need the ability to iterate quickly over massive volumes of data.
- **Language Extensions.** Some works have proposed extensions to classical languages to provide simple operations for parallel and pipeline computations, usually with special purpose optimizers, to be used on top of the core frameworks. The most representatives in this category are FlumeJava proposed by Google [17] and LINQ (Language INtegrated Query) [18]. FlumeJava is a Java library for developing and running data-parallel pipelines on top of MapReduce, as a regular single-process Java program. LINQ embeds a statically typed query language in a host programming language, such as C#, providing SQL-like construct and allowing a hybrid of declarative and imperative programming.

– **Declarative Query Languages.** In this category, we classify those languages also built on top of the core frameworks with intermediate to advanced optimizations, which compile declarative queries into MapReduce-style jobs. The most popular focused on this use-case are HiveQL of Facebook [19], Tenzing of Google [13], SCOPE of Microsoft [20,21], Spark SQL [22] on top of Spark framework, and Cheetah [23]. These languages are suitable for reporting and analysis functionalities. However, since they are built on, it is hard to achieve interactive query response times, with them. Besides, they can be considered unnatural and restrictive by programmers who prefer writing imperative scripts or code to perform data analysis.

3 Generic Architecture for Analytical Approaches

Basically, a Big Data analytical approach architecture is composed of four main components: Data Sources, Data Ingestion module, Analytic Processing module, and Analytical Requests module. A generic representation of this architecture is shown in Fig. 1 and described as follows:

– **Data Sources:** They constitute the inputs of an analytical system and often consider heterogeneous (i.e., structured, semi-structured, and unstructured) and sparse data (e.g., measures of sensors, log files streaming, mails and documents, social media content, transactional data, XML files).
– **Data Ingestion Module.** This module is in charge of moving data structured and especially unstructured data from their sources, into a system where it can be stored and analyzed. Data ingestion may be continuous or asynchronous, and can be processed in real-time, batched, or both (like lambda architectures [24]) depending upon the characteristics of the source and the destination. Most of the time, sources and destinations have not the same data timing, format, and protocol; hence, transformations or conversions are required to be usable by the destination system. Other functionalities can be available at this module, like data cleaning, filtering, aggregation, and integration. There exist many tools that can be integrated in the Data Ingestion module like Pentaho Data Integration (DI)[4], Talend[5], Apache Flume [25], Kafka [26], Sqoop [27], and Scribe [28]. Pentaho DI and Talend enable users to ingest, diverse data from any source, providing support for Hadoop distributions, Spark, NoSQL data stores, and analytic databases. Pentaho DI also allows integrating advanced analytic models from R, Python, and Weka to operationalize predictive models. Apache Flume is a distributed, reliable system for efficiently collecting, aggregating, and moving large amounts of log data from many different sources to a centralized data store. Kafka, originally developed by LinkedIn, is an open-source, fast, scalable, durable, and fault-tolerant publish-subscribe messaging system, working very well in combination with Apache Storm and Apache HBase for instance. Apache Sqoop

[4] http://www.pentaho.com/product/data-integration.
[5] http://www.talend.com.

tool is designed for transferring bulk data between HDFS and structured data stores as RDBMS. Scribe is an open-source tool, initiated by Facebook and designed for aggregating log data streamed in real-time from a large number of servers.

– **Analytic Processing Module.** It constitutes the hard core studied within this paper and can be defined as the chosen approach for the analysis phase implementation in terms of data storage (i.e., disk or memory) and the data model used (e.g., relational, NoSQL). Then, several criteria, such as parallel programming model (e.g., MapReduce, PACT) and scalability allow comparing approaches in the same class. This module is essentially composed of at least three layers:
  - Data Storage Layer. Considering an analytical approach, this layer describes whether data will be persisted locally, placed in the cloud, or cached in-memory, into a specific data model, such as a file system, a NoSQL repository, a parallel RDBMS, or a graph database.
  - Programming Model Layer. This layer represents the specific programming model (i.e., MapReduce, PACT, "in-memory") and the corresponding processing frameworks (e.g., Haddop, Storm, Spark, Nephle/Pact, Flink) that can be used. Depending on the applications needs, its priorities in terms of latency and throughput, and given the specifications of each analytic solution, a programming model and a processing framework are more suitable rather than another.
  - Data Analysis Layer. This layer can contain one or more analysis tools or "blocks" of different types:

    * **Machine Learning (ML).** This block contains libraries or toolkits facilitating learning tasks to analysts and statisticians who do not have advanced programming skills. Apache Mahout[6] and MLlib[7] are the most known ML libraries. Apache Mahout is designed for easily creating an environment for building perform and scalable ML applications. MLlib is the Apache Spark's scalable ML Library and covers the same range of learning categories as Mahout, but additionally offers the regression models, which lack to Mahout. Furthermore, relying on Spark's iterative batch and streaming approach and the use of in-memory computation, makes MLlib running jobs faster than Mahout. This advantage can be seen as inconvenient, considering the fact that MLlib is tied to Spark and cannot be performed on multiple platforms. There exist others ML frameworks like Distributed Wekahttps[8], H2O[9], Oryx[10], SAMOA[11].
    * **Scripting.** This block presents high level abstraction hiding some complexity and simplifying the programming process. For instance,

6 http://mahout.apache.org/.
7 https://spark.apache.org/mllib/.
8 http://www.cs.waikato.ac.nz/ml/index.html.
9 https://www.h2o.ai/h2o/.
10 http://oryx.io.
11 https://samoa.incubator.apache.org.

Pig [15] offers a scripting language supporting UDF written in Python, Java, JavaScript, and Ruby. Cascading[12] simplifies the development of data-intensive applications and defines a Java API for defining complex data flows and integrating those flows with back-end systems, it presents also a query planner for mapping and executing logical flows onto a computing platform.

* **Query Language/Engine.** This block represents the query language and its respective query engine, as the ones described in Sect. 2.2, that can be used in an analytic solution.

* **OLAP.** This component refers to the existence of OLAP operations and features implemented on the top of the underlying system.

* **Search Engine.** Search engines find matching items according to a set of criteria specified by the user. Tools in this block not only ensure search, but also navigation and discovery of information. Elasticsearch[13] and Solr[14] are the most known open source search engines.

* **Graph analytics.** This component presents several API developed for Graph and Graph-parallel computation, built on top of the analytic processing layer. Some examples of this kind of tools are Haloop [29], which allows iterative applications to be assembled from existing Hadoop programs without modification and significantly improves their efficiency; PageRank [30], a graph analysis algorithm that assigns weights (ranks) to each vertex by iteratively aggregating the weights of its inbound neighbors; GraphX [31] is an embedded graph processing framework built on top of Apache Spark for distributed graph computation, it implements a variant of the popular Pregel [32] interface as well as a range of common graph operations.

– **Analytical Requests.** This module considers Visualization, Reporting, Business Intelligent (BI), and Dashboards Generation functionalities. Based on the analytic data and through different query languages (e.g., procedural, SQL-like), this module can generate outputs in several formats with pictorial or graphical representation (dashboards, graphics, reports, etc.), which enables decision makers to see analytics presented visually.

We focus in the Analytics Processing module to classify the different Big Data analytic approaches.

3.1 Architecture for Analytic Processing Classification

Concerning the Analytic Processing component, several architectures exist in the Big Data context. At the beginning of the Big Data era, research studies were focused on the scalability and volume aspects of the gathered data. Thus, the data model (i.e., NoSQL, relational, graph) was an important decision to

[12] http://www.cascading.org.
[13] https://www.elastic.co/fr/products/elasticsearch.
[14] http://lucene.apache.org/solr/.

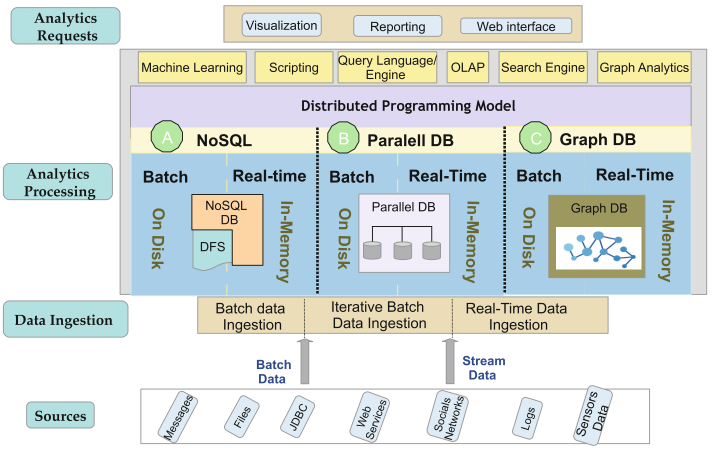


Fig. 1. Generic architecture for a Big Data Analytical approach.

implement Big Data solutions. Nowadays, due to the massive growth in areas such as social networks, cyber physical systems, smart buildings, and smart cities, the attention is diverted on the data velocity aspect. Facing the streaming data generated continually, the need of real-time analytics emerge to comply different applications requirements that demand immediate responses. Hence, tools supporting real-time issues now compete with batch solutions.

For this reason, the first aspect that we consider to classify the analytic approaches is related to the data model, as it is shown in our reference architecture depicted in Fig. 1. Specifically we have defined NoSQL based, relational parallel database based, and graph based classes. Then, each class is divided into two groups according to its analytical processing mode, batch or real-time, and consequently defining the need of a persistent disk storage or an in-memory storage, respectively. There exist hybrid architectures considering both batch and real-time processing, that have been called lambda architectures [24] and consist of three layers: Speed Layer (to process stream data), Batch Layer (to process batch data), and Serving layer (to merge both modes and present separate and common views). We find the lambda architecture included in our architecture by considering batch and real-time blocks (on any class) as lambda batch and stream layers, respectively, while the lambda server layer can be constituted by some blocks of our Data Analysis Layer (i.e., ML, Scripting, OLAP, Query Engine, Search engine, Graph Analytics). Other systems integrate both analytical and classical transactional applications in a same engine called NewSQL analytical architecture, in order to provide both batch and real-time processing. We detail each class as follows:

A- **NoSQL based Architecture.** This class of architecture uses NoSQL database model, relying on DFS (Distributed File System) or not. It generally relies on the parallel programming model offered by MapReduce, in which the processing workload is spread across many commodity compute nodes. The data is partitioned among the compute nodes at run time and the underlined framework handles inter-machine communication and machine failures. This kind of architecture is designed by the class A in Fig. 1.

B- **Relational Parallel Database based Architecture.** It is based, as classical databases, on relational tables stored on disk. It implements features like indexing, compression, materialized views, I/O sharing, caching results. Among these architectures there are: shared-nothing (multiple autonomous nodes, each owning its own persistent storage devices and running separate copies of the DBMS), shared-memory or shared anything (a global memory address space is shared and one DBMS is present), and shared-disk (based on multiple loosely coupled processing nodes similar to shared-nothing, but a global disk subsystem is accessible to the DBMS of any processing node). However, most of current analytical DBMS systems deploy a shared-nothing architecture parallel database. In this architecture, the analytical solution is based on the conjunction of the parallel (sharing-nothing) databases with a parallel programming model. This architecture is designed by class B in Fig. 1.

C- **Graph Database based Architecture.** Facing the data generation growth, there is an emerging class of inter-connected data, that can be accumulative or variable over time, on which it is necessary novel analytics – both over the network structure and across the time-variant attribute values –. Graph structure is more appropriate for this class of data. Large graphs appear in a wide range of computational domains, thus new infrastructure and programming model challenges for managing and processing this graphs emerge. The compromise between high throughput (for the offline graph analytics) and low latency (for processing high velocity graph structured streaming data) is the challenge of graph applications. We describe in the next section graph-based analytics tools, ones that process in batch-mode and others more recent graph-based tools dealing with real-time analytic processing of streaming data. We can see this type of architecture designed by class C in Fig. 1.

3.2 Criteria of Comparison

To compare the different approaches on each class, we establish a set of criteria related to the processing and interaction facilities (e.g., ML support, OLAP-like support, query languages used, Cloud services support), as well as implementation criteria related to performance aspects (e.g., scalability, programming model, fault tolerance support).

- **OLAP Support.** Some analytical solutions integrate OLAP in their system allowing operators for mining and analyzing data directly over the data analytic support (i.e., without extraction). For solutions evaluation, this property is fixed to `integrated` when OLAP is integrated or `not-integrated` otherwise.

- **Query Languages.** This criterion specifies the language on which the user relies on for querying the system. It could be `procedural`, `language extension`, or `declarative` (see Sect. 2.2).
- **Cloud Services Support.** The three types of architectures described above can be offered as a product (with `no` cloud support) or as a service deployed in the cloud. In the case of cloud support, it can be done partially, such as Infrastructure as a Service (`IaaS`, which provides the hardware support and basic storage and computing services) and Data Warehouse as a Service (`DWaaS`), or as a whole analytical solution in the cloud, i.e., Platform as a Service (`PaaS`), which provides the whole computing platform to develop Big Data analytical applications.
- **Scalability.** This criterion measures the availability on demand of compute and storage capacity, according to the volume of data and business needs. For this property in the evaluation of solutions, we fix the values `large-scale` or `medium-scale` (for the case where systems do not scale to thousands of nodes).
- **Fault Tolerance.** This criterion establishes the capability of providing transparent recovery from failures. In the context of analytical workloads, we consider the following fault tolerance techniques: the classical `Log-based` (i.e., write-ahead log), which is used for almost every database management system and `Hadoop-based` techniques, which provides data replication. In the latter, we consider two types of recovery techniques: `Hadoop-file`, where the data replication recovery control is implemented at the HDFS level (i.e., on the worked nodes allowing re-execute only the part of systems containing the failed data) and `Hadoop-programming`, where the recovery control is implemented at the model program level.
- **Programming Model.** It precises for each item which programming model is adopted. We consider `MapReduce`, `PACT`, `in-memory` models and we mention `ad-hoc` for specific implementations.
- **Machine Learning.** Since ML techniques are currently used in many areas including commerce, finance, healthcare, and entertainment, it is nowadays associated to the Big Data dilemma and can be coupled to the analytical aspect. Taking into account ML facility support, some analytic solutions can be more attractive to users rather than another. For solutions evaluation, this property is fixed to `yes` when a library/API exists and was planned to be combined to the solution or `no` otherwise.

4 Describing Some Big Data Analytic Systems

In this section we describe some well-known frameworks/systems for analytic processing on the three classes of architectures.

4.1 NoSQL Based Architectures

Several tools which are NoSQL based architecture exist. Some of them execute the analysis of the data off-line, thus they are classified as **batch solutions**. Most popular tools in this classification are:

- **Avatara.** It leverages an offline elastic computing infrastructure, such as Hadoop, to precompute its cubes and perform joins outside of the serving system [33]. These cubes are bulk loaded into a serving system periodically (e.g., every couple of hours). It uses Hadoop as its batch computing infrastructure and Voldemort [34], a key-value storage, as its cube analytical service. The Hadoop batch engine can handle terabytes of input data with a turnaround time of hours. While Voldemort, as the key-value store behind the query engine, responds to client queries in real-time. Avatara works well while cubes are small, for far bigger cubes, there will be significant network overhead in moving the cubes from Voldemort for each query.
- **Apache Kylin**[15]**.** It is a distributed analytic engine supporting definition of cubes, dimensions, and metrics. It provides SQL interface and OLAP capability based on Hadoop Ecosystem and HDFS environment. It is based on Hive data warehouse to answer mid-latency analytic queries and on materialized OLAP views stored on a HBase cluster to answer low-latency queries.
- **Hive.** It is an open-source project that aims at providing data warehouse solutions and has been built by the Facebook Data Infrastructure Team on top of the Hadoop environment. It supports ad-hoc queries with a SQL-like query language called HiveQL [19]. These queries are compiled into MapReduce jobs that are executed using Hadoop. The HiveQL includes its own system type and Data Definition Language (DDL) which can be used to create, drop, and alter tables in a database. It also contains a system catalog which stores metadata about the underlying table, containing schema information and statistics, much like DBMS engines. Hive currently provides only a simple, naive rule-based optimizer.
- **Cloudera Impala**[16]**.** It is a parallel query engine that runs on Apache Hadoop. It enables users to issue low-latency SQL queries to data stored in HDFS and Apache HBase without requiring data movement or transformation. Impala uses the same file and data formats, metadata, security, and resource management frameworks used by MapReduce, Apache Hive, Apache Pig, and other Hadoop software. Impala allows to perform analytics on data stored in Hadoop via SQL or business intelligence tools. It gives a good performance while retaining a familiar user experience. By using Impala, a large-scale data processing and interactive queries can be done on the same system using the same data and metadata without the need to migrate data sets into specialized systems or proprietary formats.

Other solutions can be classified as **NoSQL based real-time solutions**, because they execute the analysis over the streaming data, some examples are:

- **Shark.** It is a data analysis system that leverages distributed shared memory to support data analytics at scale and focuses on in-memory processing of analysis queries [22]. It supports both SQL query processing and ML functions. It is built on the distributed shared memory RDD abstraction from

[15] http://kylin.apache.org.
[16] https://www.cloudera.com/products/open-source/apache-hadoop/impala.html.

Spark, which provides efficient mechanisms for fault recovery. If one node fails, Shark generates deterministic operations necessary for building lost data partitions in the other nodes, parallelizing the process across the cluster. Shark is compatible with Apache Hive, thus it can be used to query an existing Hive data warehouse and to query data in systems that support the Hadoop storage API, including HDFS and Amazon S3.

– **Mesa.** Google Mesa [35] leverages common Google infrastructure and services, such as Colossus (the successor of Google File System) [36], BigTable [37], and MapReduce. To achieve storage scalability and availability, data is horizontally partitioned and replicated. Updates may be applied at the granularity of a single table or across many tables. To ensure consistent and repeatable queries during updates, the underlying data is multi-versioned. To achieve update scalability, data updates are batched, assigned a new version number, and periodically (e.g., every few minutes) incorporated into Mesa. To achieve update consistency across multiple data centers, Mesa uses a distributed synchronization protocol based on Paxos [38].

Discussion and Comparison. NoSQL-based systems have shown to have superior performance than other systems (such as parallel databases) in minimizing the amount of work that is lost when a hardware failure occurs [8,39]. In addition, Hadoop (which is the open source implementations of MapReduce) represents a very cheap solution. However, for certain cases, MapReduce is not a suitable choice, specially when intermediate processes need to interact, when lot of data is required in a processing, and in real time scenarios, in which other architectures are more appropriate.

4.2 Relational Parallel Databases Based Architectures

Several projects aim to provide low-latency engines, whose architectures resemble shared-nothing parallel databases, such as projects which embed MapReduce and related concepts into traditional parallel DBMSs. We classify as parallel databases **batch solutions**, those that execute the analysis of the data off-line. Some examples are:

– **PowerDrill.** It is a system which combines the advantages of columnar data layout with other known techniques (such as using composite range partitions) and extensive algorithmic engineering on key data structures [40]. Compared to Google's Dremel that uses streaming from DFS, PowerDrill relies on having as much data in memory as possible. Consequently, it is faster than Dremel, but it supports only a limited set of selected data sources, while Dremel supports thousands of different data sets. PowerDrill uses two dictionaries as basic data structures for representing a data column. Since it relies on memory storage, several optimizations are proposed to keep small the memory footprint of these structures. PowerDrill is constrained by the available memory for maintaining the necessary data structures.

- **Teradata.** It is a system which tightly integrates Hadoop and a parallel data warehouse, allowing a query to access data in both stores by moving (or storing) data (i.e., the working set of a query) between each store as needed [41]. This approach is based on having corresponding data partitions in each store co-located on the same physical node. Thus reducing network transfers and improving locality for data access and loading between the stores. However, this requires a mechanism whereby each system is aware of the other systems partitioning strategy; the partitioning is fixed and determined up-front. Even though these projects offer solutions by providing a simple SQL query interface and hiding the complexity of the physical cluster, they can be prohibitively expensive at web scale.
- **Microsoft Azure**[17]**.** It is a solution provided by Microsoft for developing scalable applications for the cloud. It uses Windows Azure Hypervisor (WAH) as the underlying cloud infrastructure and .NET as the application container. It also offers services including Binary Large OBject (BLOB) storage and SQL service based on SQL Azure relational storage. The analytics related services provide distributed analytics and storage, as well as interactive analytics, big data analytics, data lakes, ML, and data warehousing.
- **AsterData System.** It is a nCluster shared-nothing relational database[18], that uses SQL/MapReduce (SQL/MR) UDF framework, which is designed to facilitate parallel computation of procedural functions across hundreds of servers working together as a single relational database [42]. The framework leverages ideas from the MapReduce programming paradigm to provide users with a straightforward API through which they can implement a UDF in the language of their choice. Moreover, it allows maximum flexibility, since the output schema of the UDF is specified by the function itself at query plan-time.
- **Google's Dremel.** It is a system that supports interactive analysis of very large datasets over shared clusters of commodity machines [43]. Dremel is based on a nested column-oriented storage that is designed to complement MapReduce and uses streaming from DFS. It is used in conjunction with MapReduce to analyze outputs of MapReduce pipelines or rapidly prototype larger computations. It has the capability of running aggregation queries over trillion-row tables in seconds by combining multi-level execution trees and columnar data layout. The system scales to thousands of CPUs and petabytes of data and has thousands of users at Google.
- **Polybase.** It is a feature of Microsoft SQL Server Parallel Data Warehouse (PDW), which allows directly reading HDFS data into databases, by using SQL [44]. It allows to reference HDFS data through external PDW tables and joined with native PDW tables using SQL queries. It employs a split query processing paradigm in which SQL operators on HDFS-resident data are translated into MapReduce jobs by the PDW query optimizer and then executed on the Hadoop cluster.

[17] http://www.microsoft.com/azure.
[18] http://www.asterdata.com/.

- **Cubrick.** It is an architecture that enables data analysis of large dynamic datasets [45]. It is an in-memory distributed multidimensional database that can execute OLAP operations such as *slice* and *dice*, *roll up*, and *drill down* over terabytes of data. Data in a Cubrick cube is range partitioned in every dimension, composing a set of data containers, called *bricks*, where data is stored sparsely in an unordered and append-only fashion, providing high data ingestion ratios and indexed access through every dimension. Unlike traditional data cubes, Cubrick does not rely on any pre-calculation, rather it distributes the data and executes queries on-the-fly leveraging MPP architectures. Cubrick is implemented at Facebook from the ground up.
- **Amazon Redshift.** It is an SQL-compliant, massively-parallel, query processing, and database management system designed to support analytics workload [46]. Redshift has a query execution engine based on ParAccel[19], a parallel relational database system using a shared-nothing architecture with a columnar orientation, adaptive compression, memory-centric design. However, it is mostly PostgreSQL-like, which means it has rich connectivity via both JDBC and ODBC and hence Business Intelligence tools. Based on EC2, Amazon Redshift solution competes with traditional data warehouse solutions by offering DWaaS, that it is translated on easy deployment and hardware procurement, the automated patching provisioning, scaling, backup, and security.
- **Nanocubes[20].** It is an in-memory data cube engine offering efficient storage and querying over spatio-temporal multidimensional datasets. It enables real-time exploratory visualization of those datasets. The approach is based on the construction of a data cube (i.e., a nanocube), which fits in a modern laptop's main memory, and then computes queries over that nanocube using OLAP operations like *roll up* and *drill down*.

Other solutions can be classified as **parallel-based real-time or NewSQL analytical solutions**, because they execute the analysis over the streaming data or transaction applications data, such as:

- **Druid.** It is a distributed and column-oriented database designed for efficiently support OLAP queries over real-time data [47]. A Druid cluster is composed of different types of nodes, each type having a specific role (real-time node, historical nodes, broker nodes, and coordinator nodes), that operate independently. Real-time nodes ingest and query event streams using an in-memory index to buffer events. The in-memory index is regularly persisted to disk. Persisted indexes are then merged together periodically before getting handed off. Historical nodes are the main workers of a Druid cluster, they load and serve the immutable blocks of data (segments) created by the real-time nodes. Broker nodes route queries to real-time and historical nodes and merge partial results returned from the two types of nodes. Coordinator nodes are essentially responsible for managing and distribute the data on

[19] http://www.actian.com.
[20] http://nanocubes.net.

historical nodes: loading new data, dropping outdated data, replicating. A query API is provided in Druid, with a new proposed query language based on JSON Objects in input and output.
- **Vertica.** It is a distributed relational DBMS that commercializes the ideas of the C-Store project [48]. The Vertica model uses data as tables of columns (attributes), though the data is not physically arranged in this manner. It supports the full range of standard INSERT, UPDATE, DELETE constructs for logically inserting and modifying data as well as a bulk loader and full SQL support for querying. Vertica supports both dynamic updates and real-time querying of transactional data.
- **SAP HANA**[21]**.** SAP HANA is an in-memory and column-oriented relational DBMS. It provides both transactional and real-time analytics processing on a single system with one copy of the data. The SAP HANA in-memory DBMS provides interfaces to relational data (through SQL and Multidimensional expressions, or MDX), as well as interfaces to column-based relational DBMS, which allows to support geospatial, graph, streaming, and textual/unstructured data. The approach offers multiple in-memory stores: row-based, column-wise, and also object graph store.
- **Flink** (See footnote 3) **(formerly known as Stratosphere).** It is an open-source framework designed for processing real-time and batch data. It contains an execution engine which includes query processing algorithms in external memory and the PACT programming model together with its associated framework Nephele. Through Nephele/PACT, it is possible to treat low level programming abstractions consisting of a set of parallelization primitives and schema-less data interpreted by the UDFs written in Java. Flink executes the analytic process directly from the data source (i.e., data is not stored before the analytical processing), by converting data to binary formats after the initial scans. Besides, it provides a support for iterative programs that make repeated passes over a data set updating a model until they converge to a solution a dataflow Graph is compiled down by all Flink Programs. Nodes represents operations (map, reduce, join or filter), edges represent the flow of data between operations. In addition, the Flink open-source community has developed libraries for machine learning and graph processing.

Discussion and Comparison. Classical parallel database systems serve some of the Big Data analysis needs and are highly optimized for storing and querying relational data. However, they are expensive, difficult to administer, and lack fault-tolerance for long-running queries [8,39]. Facing the Big Data generating growth, new relational database systems have been designed to support fault tolerance through replication, flow control, and distributed query processing, providing the same scalable performance of NoSQL databases. Some of these systems try to facilitate parallel computation across hundreds of servers either by integrating parallel databases and frameworks such as Hadoop or SQL/MR-UDF (e.g., Teradata, AsterdData), or by using SQL directly over a DFS (e.g.,

[21] https://help.sap.com/viewer/product/SAP_HANA_PLATFORM/2.0.00/en-US.

Polybase, Dremel). Others also support real-time data analysis (e.g., Druid and SAP HANA). HANA support transactional real-time data analysis, however Druid support streaming data. By the other side, SAP-HANA simultaneously also supports classical transactional processing. Cubrick, Druid, and SAP HANA are in-memory column-oriented relational DBMS and Vertica is a disk-store system. Teradata, Vertica, Cubricks, Druid, Nanocubes, and Flink (Stratosphere) have in common the support for OLAP operations. Cubrick is distinguished by the execution on the fly, without pre-calculations and multidimensional indexing techniques, supporting ad-hoc query, which is not provided for most current OLAP DBMSs.

4.3 Graph Based Architectures

In the literature, several works propose distributed graph frameworks with functionalities on fast graph parallel computation, exploration, and querying with specific graph programming model, in most cases they are called vertex-centric systems. Graph-based solutions that execute the analysis of the data off-line are classified as **batch solutions**. Most popular tools in this classification are:

- **Pregel** [32]**.** It is a Large-scale graph computing distributed programming framework, based on the Bulk Synchronous Processing (BSP) [49]. Pregel, developed by Google, provides a natural API for programming graph algorithms while managing the details of distribution, messaging, and fault tolerance. Pregel expresses naturally computation dependencies. It offers a vertex-centric graph programming abstraction that uses a BSP model with individual tasks being composed as if operating independently on a single vertex. Each barrier synchronization step is called a super-step and vertices distributed across a cluster exchange through messages with neighbors at these synchronization boundaries. The distributed graph programming model so built, is simple but elegant, similar to MapReduce. However, in practice, the barriers are costly and the number of super-steps required can be large depending on the type of algorithm. Source code of the Pregel project was not made public.
- **Apache Giraph**[22]**.** It was designed to bring large-scale graph processing to the open source community, based loosely on the Pregel model, while providing the ability to run on existing Hadoop infrastructure. Giraph adds several functionalities compared to the basic Pregel model (master computation, shared aggregators, edge-oriented input, out-of-core computation, etc.). Apache Giraph is currently used at Facebook to analyze the social graph formed by users and their connections.
- **GraphLab** [50]**.** It expresses asynchronous, dynamic, graph-parallel computation, while ensuring data consistency. It also achieves a high degree of parallel performance in a persistent shared-memory model, by setting a sequential shared memory abstraction to several vertices. Each vertex can read and write to data on adjacent vertices and edges. The GraphLab runtime is then

[22] Apache Giraph - http://giraph.apache.org.

responsible for ensuring a consistent parallel execution. Originally, GraphLab was developed for ML tasks, but it proves to be efficient at a broad range of other data-mining tasks.

- **Goffish** [51]**.** It is a Sub-Graph Centric Framework for Large-Scale Graph Analytics co-designed with a distributed persistent graph storage for large scale graph analytics on commodity clusters, offering the added natural flexibility of shared memory sub-graph computation. Goffish is composed essentially of two components: the GoFS distributed storage layer for storing time series graphs, and the Gopher subgraph-centric programming model for imposing and executing graph on-line analytics. Gopher is based on the scalable vertex centric BSP like Pregel, Giraph, and GraphLab. They are analogous to the HDFS file system and MapReduce programming model for Hadoop, but have been designed and developed with a focus on timeseries graph analytics. Goffish proposes a distributed storage model optimized for time-series graphs called Graph-oriented File System (GoFS). GoFS partitions the graph based on topology, groups instances over time on disk, and allows attribute-value pairs to be associated with the vertices and edges of the instances. In particular, GoFS is designed to scale out for common data access patterns over time-series graphs, and this data layout is intelligently leveraged by Gopher during execution.

Other solutions can be classified as **graph-based real-time solutions**, because they execute the analysis over the streaming data, such as:

- **Trinity** [52]**.** It is a general purpose distributed graph system over a memory cloud. Trinity offers efficient parallel computing and support of fast graph exploration thanks to network communication and optimized memory management. Memory cloud is a globally addressable, in-memory key-value stored over a cluster of machines. Trinity supports both low-latency on-line query processing and high-throughput off-line (batch) analytics on billion-node. It is considered in our classification as batch processing, from analytics point of view.
- **GraphCEP** [53]**.** It is a real-time data analytics using parallel complex event and graph processing. Complex event processing (CEP) [54] systems allows the detection of multiple instances of a queried pattern on an incoming streams in parallel. Streams are divided into partitions processed in parallel by an elastic number of operator instances. The idea behind GraphCEP is to combine Parallel CEP to graph processing in order to bring a streaming graph-structured data analysis solution.
- **Microsoft Graph Engine (GE)**[23]**.** It is based on Trinity, improved as an open-source, distributed, in-memory, large graph processing engine. It relies on a strongly-typed RAM store and a general distributed computation engine. Data can be inserted into GE and retrieved at high speed since it is kept in-memory and only written back to disk as needed. GE is a graph data management system designed for managing real-life graphs with rich associated data.

[23] Graph Engine https://www.graphengine.io.

Discussion and Comparison. We notice that Pregel, Apache Giraph, and GraphLab have the use of Bulk Synchronous Processing in common. Practical differences can be emphasized. Pregel is closed source which makes it unopened to evolution and integration. GraphLab is written in C++ and make its interoperability with existent Hadoop infrastructure tedious and time consuming. Giraph is written in Java and has vertex and edge input formats that can access MapReduce input formats. Users can insert Giraph applications into existing Hadoop pipelines and leverage operational expertise from Hadoop. However, GraphLab do not scale at Facebook data size for instance with over 1.39B users and hundreds of billions of social connections. Trinity offers a shared memory abstraction in a distributed memory infrastructure. Algorithms use both message passing and a distributed address space called memory cloud. However, this assumes large memory machines with high speed interconnects. GraphCE combines the scalability and the efficiency of distributed graph processing systems and the expressiveness and timeliness of CEP and performs better results than those of CEP or Graph systems.

4.4 General Discussion

Traditional BI operations are oriented to perform analytical queries and ML algorithms on historical data. Thus data warehouses were focused on executing complex read-only queries (i.e., aggregations, multiway joins) that take a long time to process large data sets (e.g., seconds or even minutes). Each of these queries can be significantly different than the previous. The first analytical solutions for the Big Data were inspired on the classical systems where generally data warehouse and OLAP systems were treated by different engines and the target distributed architectures. On the other side, originally, neither HDFS nor MapReduce were designed with real-time performance in mind, thus real-time treatments required move data out from Hadoop to another system able to deliver real-time processing In order to improve these systems, some solutions have been proposed coupling in-memory file systems, such as Ignite system[24] and HDFS, allowing to deliver real-time processing without moving data out of Hadoop onto another platform. Others systems have integrated in only one engine in charge of managing both the historical data for batch processing and the new data for real-time processing.

We summarize our classification in Table 1 and compare the revised approaches according to the established criteria: used query language, scalability, OLAP support, fault tolerance support, cloud support, programming model, and ML support. Generally, solutions in the class *A* offer materialized views stored on NoSQL databases and updated in short period of times to offer OLAP facilities, meanwhile for in-memory based architectures the OLAP facility is embedded in the system. Approaches based on parallel databases do not scale to huge amount of nodes, however they have generally, best performance than the other architectures. Most languages used are declarative SQL-compliant. Regarding fault

[24] https://ignite.apache.org.

tolerance, most Big Data approaches leverage on Hadoop facilities, instead of the classical log-based mechanism. Currently, Big Data analytical solutions trend to move to the cloud, either partially or as whole solution (PaaS). Concerning the parallel programming model, most solutions described in this work implement MapReduce programming model native with Hadoop as the underline support or were extended in order to consider this programming model. Other solutions have extended the MapReduce programming model such Cubrick and Shark that also support Spark modeling with RDD.

New applications, besides perform analysis on historical data like traditional BI operations, need to support an analysis of the historical data aggregated with tackled data items as they arrive by the streaming. In this context, several solutions have been proposed. We can distinguished two groups treating real-time differently:

- **Lambda Architectures:** represented by systems which separate batch processing system (e.g., Hadoop, Spark) to compute a comprehensive view on historical data, while simultaneously use a stream processing system (e.g., Storm, Spark) to provide views of incoming data. It requires the application developer to write a query for multiple systems if they want to combine data from different databases. Flink from the class B or Shark from the class A are examples of Lambda Architecture.
- **Transactional-Analytical Hybrid Architectures**: represented by systems that integrate execution engines within a single DBMS to support OLTP (On line Transactions Processing) with high throughput and low latency demands of workloads, while also allowing for complex, longer running OLAP queries to operate on both new data (fresh data or transactional data) and historical data. SAP HANA, Teradata and Vertica belonging to the class B, are examples of this kind of architecture.

The Lambda architecture gives a fresh views by analysis compiled on-line, with a possibility to generate batch or real-time views separately, but needs the management of different frameworks (for batch and real-time). Transactional-Analytical Hybrid architectures offers faster OLAP analytics, with higher concurrency and the ability to update data as it is being analyzed.

Graph-based architectures may interest some specific context and domain where data are closely connected with explicit relationship between them and where modeling data in a graph structure is more suitable. Even though graph-based solutions have proven their performance, as well in batch mode as real-time processing, migration to graph systems, when the existing system is relational for instance, may be time-consuming, costly.

5 Related Work

General comparative studies of the most popular Big Data frameworks are presented in extensive surveys conducted to discuss Big Data frameworks for analytics [55,56]. Authors base the comparison on a list of qualitative criteria, some

Table 1. Comparative table of the studied analytic Big Data approaches.

Architecture	**Tool**	**Query Language**	**Scalability**	**OLAP**	**Fault Tolerance**	**On the cloud**	**Programming model**	**ML**
NoSQL based Architectures (Class *A*)	Avatara	Language extensions	Large-scale	integrated	Hadoop-programming	no	MapReduce	no
	Hive	Declarative (HiveQL)	Large-scale	not-integrated	Hadoop-programming	DWaaS	MapReduce	yes Hivemall
	Cloudera Impala	Declarative (HiveQL) and Procedural (PIG Latin)	Large-scale	not-integrated	Hadoop-programming	DWaaS	MapReduc	yes Cloudera Oryx
	Apache Kylin	Language extensions	Large-scale	integrated	Hadoop-programming	no	MapReduce	no
	Mesa	Language extension	Large-scale	not-integrated	Hadoop-file	DWaaS	MapReduce	no
	Shark	Declarative (HiveQL)	Large-scale	not-integrated	Hadoop-file (RDD properties on Spark)	no	RDD-Spark	yes
Relational Parallel Database based Architectures (Class *B*)	PowerDrill	Procedural	Medium-scale	not-integrated	Log-based	no	ad-hoc	no
	Teradata	Procedural	Medium-scale	not-integrated	Hadoop-file	DWaaS	MapReduce	yes ThinkDeep
	Microsoft Azure	Declarative (SQL Azure)	Large-scale	not-integrated	Hadoop-file and Hadoop-programming	IaaS PaaS	MapReduce	yes Azure ML
	AsterData	Declarative	Medium-scale	not-integrated	Hadoop-file	IaaS	MapReduce	no
	Google's Dremel	Procedural	Large-scale	not-integrated	Log-based	IaaS (BigQuery)	MapReduce	no
	Amazon Redshift	Declarative	Large-scale	not integrated	Hadoop-file	PaaS, DWaaS	MapReduce	no
	Vertica	Declarative	Medium-scale	not-integrated	Hadoop-file	no	MapReduce	no
	Cubrick	Declarative	Large-scale	integrated	Log-based	no	ad-hoc	no
	Druid	Procedural (based on JSON)	Large-scale	integrated	Log-based	no	ad-hoc	no
	SAP HANA	Declarative	Large-scale	integrated	Log-based	PaaS	ad-hoc	yes PAL
	Nanocubes	Declarative	Large-scale	integrated	Log-based	no	ad-hoc	
	Flink	Declarative (Meteor)	Large-scale	not-integrated	Log-based	IaaS	Nephele/PACT	yes FlinkML21
Graph based Architectures (Class *C*)	Pregel	Procedural	Large-scale	not-integrated	Log-based	no	ad-hoc	no
	Giraph	Declarative	Large-scale	not-integrated	Hadoop-file	no	MapReduce	no
	GraphLab	Procedural	Large-scale	not-integrated	Log-based	yes	ad-hoc	yes
	Goffish	Procedural	Large-scale	not-integrated	Log-based	yes	ad-hoc	no
	Trinity	Procedural	Large-scale	not-integrated	Log-based	yes	ad-hoc	no
	GraphCEP	Procedural	Large-scale	not-integrated	Log-based	no	ad-hoc	no
	Graph Engine	Procedrual	Large-scale	not-integrated	log-based	no	ad-hoc	no

of them similar to our proposal (e.g., fault tolerance support, scalability, programming model, supported programming language). Some of them also show experiments to evaluate performance, scalability, and the resource usage. These studies are useful to evaluate general advantages and drawbacks of the analytics tools, however, none of these surveys propose a generic classification of those tools.

Some other recent studies present comparative analyses of some aspects in the context of Big Data [57–59]. Authors in [57] present a comparative study between two cloud architectures: Microsoft Azure and Amazon AWS Cloud services. They consider price, administration, support, and specification criteria to compare them. The work presented in [58] proposes a classification of technologies, products, and services for Big Data Systems, based an a reference architecture representing data stores, functionality, and data flows. For each solution, they present a mapping between the use case and the reference architecture. The paper studies data analytics infrastructures at Facebook, LinkedIn, Twitter, Netflix, and BlockMon. A set of criteria to compare ML tools for Big Data is presented in [59]. Several ML frameworks are evaluated in the context of several processing engines (e.g., Spark, MapReduce, Storm, Flink) according to their scalability, speed, coverage, usability, and extensibility. These criteria support the decision of selection an appropriate ML tool, to users that have knowledge on ML, by providing as much information as possible and quantifying what the tradeoffs will be for learning from Big Data. However, these works are specific systems-concentrated and do not provide a general vision of existing approaches.

The work presented in [60] is the most related to our paper. It aims to define the six pillars (namely Storage, Processing, Orchestration, Assistance, Interfacing, and Development) on which Big Data analytics ecosystem is built. For each pillar, different approaches and popular systems implementing them are detailed. Based on that, a set of personalized ecosystem recommendations is presented for different business use cases. Even though authors argue that the paper assists practitioners in building more optimized ecosystems, we think that proposing solutions for each pillar does not necessarily imply a global coherent Big Data analytic system. Even though our proposed classification considers building blocks on which Big Data analytics Ecosystem is built (presented as layers in the generic architecture), our aim is to provide the whole view of the analytical approaches. Practitioners can determine which analytical solution, instead of a set of building blocks, is the most appropriate according to their needs.

6 A Decision Support System to Select Big Data Analytics: A Perspective

In this section we propose a general idea of a Decision Support System (DSS) that based on our proposed architecture can recommend an appropriate classification of Big Data analytic approach according to the specific case of the user. The proposed architecture of our DSS is inspired on the integrated model for decision making described in [61]. Even though the approach presented in that work is focused on DSS that use Big Data techniques to support decision-making processes in organizations, we think that the proposed model can be adapted for a DSS to select Big Data analytic tools. Also, we have revised other works proposed to support the selection of DBMS [62] and cloud databases [63], which use ontologies and ranking approaches to generate, evaluate, and compare candidates to select.

Authors in [61] define a DSS as interactive, computer-based Information Systems that help decision makers utilize data, models, solvers, visualizations, and the user interface to solve semi-structured or unstructured problems. Selecting Big Data tools can be considered as a semi-structured problem, as long as we can formally describe the Big Data approaches, the user use case, and criteria, based for example on ontologies. Authors state that the process of decision-making (also called Business Intelligence – BI) is marked by two kinds of elements: organizational and technical. The organizational elements are those related to the daily functioning of companies where decisions must be made and aligned with the companies strategy. In our case, we can consider that these elements can be extracted from the specific use case or scenario presented by the user. Also, as authors suggest, it can be centered on data analytics over the data collected from previous experiences and the feedback obtained from users. The technical elements include the toolset used to aid the decision making process such as information systems, data repositories, formal modeling, analysis of decision. It is focused on tools and technologies for data storage and mining for knowledge discovery. These elements are the most important in our work.

The main objective of a DSS is to support a decision by determining which alternatives to solve the problem are more appropriate. In our case, the alternatives are defined by our proposed classification: *which Big Data analytic classification is the most appropriate for the specific use case of the user*. According to the authors of [61], DSS have a set of basic elements that includes a data base and a model base with their respective management, the business rules to process data according a chosen model, and a user interface. Data and model bases and their respective management system allow for business rules in processing data according to a model to formulate the possibilities of solutions for the problem. In our case, the data base are the registered experiences obtained from users, the model base is our proposed classification architecture, the business rules are defined based on our proposed set of criteria for evaluation. Thus, based on the integrated model proposed in [61], we propose a similar architecture for our DSS, as shown in Fig. 2. The *Content acquisition* can be done through private data and public data sources. The private data source is managed and produced by the DSS and represents the users experiences, users feedback, successfully scenarios, etc., registered in our system in structured repositories. The public data represents experiences not registered in the system, opinions, texts, videos, photos, etc., that can be unstructured data. In contrast to the proposal in [61], regarding the use of *Business Intelligence* to generate alternatives, we pretend to use *multi-criteria decision making* methods as in [62,63] to rank the classification candidates in terms of our list of criteria of evaluation and the criteria weights provided by users.

The *Generation of alternatives* in terms of our proposed criteria. The front end of the DSS should offer different visualization, reports, and recommendations in a user-friend interface. Finally, Feedback from users, learning from the user selection alternatives and use cases presented.

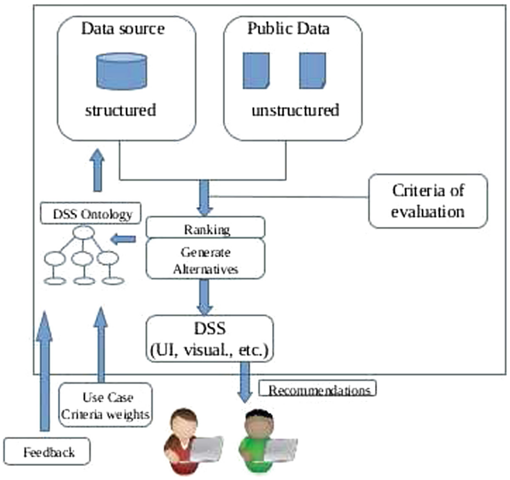


Fig. 2. Architecture of the decision support system.

We also pretend to define a core ontology for our DSS to facilitate the generation, ranking, and evaluation of candidates based on user requirements and the list of criteria of comparison. The ontology should allow to model the different proposed classifications (i.e., provide concepts related to Big Data analytic tools), the list of criteria of comparison (i.e., concepts that describe how to define and evaluate each criterion of comparison), and the use cases (i.e., concepts that provide facilities to users to model their use cases), as well as to model users feedback (i.e., concepts related to the evaluation of the experiences of users with Big Data analytics tools).

The user specifies its use case and the criteria weights for the multi-criteria decision making method, through the user interface, by using for example an interactive process or an XML-like or JSON-like language. Then, the DSS automatically generates the model of the use case based on the ontology.

7 Conclusions

Social networks, Internet of Things and cyber physical systems constitutes sources of huge amount of generated data. Several Big Data Analytical approaches exist with different architectures. We address in this paper the classification of Big Data analytical approaches. We propose and present a generic architecture for Big Data analytical approaches allowing to classify them according to the data storage layer: the type of database used and the data model and the processing mode: real-time or batch. We focus on the three most recently used architectures, as far as we know: NoSQL-based, Parallel databases based, and graph based architectures. We compare several implementations based on criteria such as: OLAP support, scalability as the capacity to adapt the volume of data and business needs, type of language supported, and fault tolerance in terms of the need of restarting a query if one of the node involved in the query processing fails. Machine Learning algorithms may be implemented through libraries built on top of the analytic processing layer, we considered this possibility as criteria of choice of an analytical tool rather than another. Finally, we present a general idea of a decision support system to select Big Data Analytic solution as perspective of our future work.

References

1. Kune, R., Konugurthi, P.K., Agarwal, A., Chillarige, R.R., Buyya, R.: The anatomy of big data computing. Softw. Pract. Exp. **46**, 79–105 (2016)
2. Grolinger, K., Higashino, W.A., Tiwari, A., Capretz, M.A.: Data management in cloud environments: NoSQL and NewSQL data stores. J. Cloud Comput.: Adv. Syst. Appl. **2**, 22 (2013)
3. Pavlo, A., Aslett, M.: What's really new with NewSQL? SIGMOD Rec. **45**, 45–55 (2016)
4. Chen, M., Mao, S., Liu, Y.: Big data: a survey. Mob. Netw. Appl. **19**, 171–209 (2014)
5. Philip Chen, C., Zhang, C.Y.: Data-intensive applications, challenges, techniques and technologies: a survey on big data. Inf. Sci. **275**, 314–347 (2014)
6. Cardinale, Y., Guehis, S., Rukoz, M.: Big data analytic approaches classification. In: Proceedings of the International Conference on Software Technologies, ICSOFT 2017, pp. 151–162. SCITEPRESS (2017)
7. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Cambridge University Press, Cambridge (2014)
8. Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., Stonebraker, M.: A comparison of approaches to large-scale data analysis. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 165–178 (2009)
9. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**, 107–113 (2008)
10. Battré, D., et al.: Nephele/PACTs: a programming model and execution framework for web-scale analytical processing. In: Proceedings of Symposium on Cloud Computing, pp. 119–130 (2010)
11. Warneke, D., Kao, O.: Nephele: efficient parallel data processing in the cloud. In: Proceedings of Workshop on Many-Task Computing on Grids and Supercomputers, pp. 8:1–8:10 (2009)

12. Zaharia, M., Chowdhury, M., Das, T., Dave, A., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of Conference on Networked Systems Design and Implementation, pp. 15–28 (2012)
13. Chattopadhyay, B., Lin, L., Liu, W., Mittal, S., et al.: Tenzing: a SQL implementation on the MapReduce framework. PVLDB **4**, 1318–1327 (2011)
14. Pike, R., Dorward, S., Griesemer, R., Quinlan, S.: Interpreting the data: parallel analysis with Sawzall. Sci. Program. **13**, 277–298 (2005)
15. Olston, C., Reed, B., Srivastava, U., Kumar, R., et al.: Pig latin: A not-so-foreign language for data processing. In: Proceedings of International Conference on Management of Data, pp. 1099–1110 (2008)
16. Beyer, K.S., Ercegovac, V., Gemulla, R., Balmin, A., Eltabakh, M.Y., et al.: Jaql: a scripting language for large scale semistructured data analysis. PVLDB **4**, 1272–1283 (2011)
17. Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R.R., Bradshaw, R., Weizenbaum, N.: FlumeJava: easy, efficient data-parallel pipelines. SIGPLAN Not. **45**, 363–375 (2010)
18. Meijer, E., Beckman, B., Bierman, G.: LINQ: reconciling object, relations and XML in the .NET framework. In: Proceedings of ACM International Conference on Management of Data, p. 706 (2006)
19. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., et al.: Hive - a petabyte scale data warehouse using hadoop. In: Proceedings of International Conference on Data Engineering, pp. 996–1005 (2010)
20. Zhou, J., Bruno, N., Wu, M.C., Larson, P.A., Chaiken, R., Shakib, D.: SCOPE: parallel databases meet MapReduce. VLDB J. **21**, 611–636 (2012)
21. Chaiken, R., Jenkins, B., et al.: SCOPE: easy and efficient parallel processing of massive data sets. VLDB Endow. **1**, 1265–1276 (2008)
22. Xin, R.S., Rosen, J., Zaharia, M., Franklin, M.J., Shenker, S., Stoica, I.: Shark: SQL and rich analytics at scale. In: Proceedings of ACM International Conference on Management of Data, pp. 13–24 (2013)
23. Chen, S.: Cheetah: a high performance, custom data warehouse on top of MapReduce. VLDB Endow. **3**, 1459–1468 (2010)
24. Hasani, Z., Kon-Popovska, M., Velinov, G.: Lambda architecture for real time big data analytic. In: ICT Innovations 2014 Web Proceedings, pp. 133–143 (2014)
25. (Apache Flume). http://flume.apache.org/
26. Wang, G., Koshy, J., Subramanian, S., Paramasivam, K., Zadeh, M., Narkhede, N., Rao, J., Kreps, J., Stein, J.: Building a replicated logging system with Apache Kafka. Proc. VLDB Endow. **8**, 1654–1655 (2015)
27. (Apache Sqoop). http://sqoop.apache.org/
28. Lee, G., Lin, J., Liu, C., Lorek, A., Ryaboy, D.: The unified logging infrastructure for data analytics at Twitter. VLDB Endow. **5**, 1771–1780 (2012)
29. Bu, Y., Howe, B., Balazinska, M., Ernst, M.D.: The HaLoop approach to large-scale iterative data analysis. VLDB J. **21**, 169–190 (2012)
30. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: bringing order to the web. In: Proceedings of the International WWW Conference, Brisbane, Australia, pp. 161–172 (1998)
31. Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: GraphX: graph processing in a distributed dataflow framework. In: Proceedings of the USENIX Conference on Operating Systems Design and Implementation, pp. 599–613 (2014)

32. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: Proceedings of the ACM International Conference on Management of Data, pp. 135–146. ACM (2010)
33. Wu, L., Sumbaly, R., Riccomini, C., Koo, G., Kim, H.J., Kreps, J., Shah, S.: Avatara: OLAP for web-scale analytics products. Proc. VLDB Endow. **5**, 1874–1877 (2012)
34. Sumbaly, R., Kreps, J., Gao, L., Feinberg, A., Soman, C., Shah, S.: Serving large-scale batch computed data with project Voldemort. In: Proceedings of the USENIX Conference on File and Storage Technologies, p. 18 (2012)
35. Gupta, A., Yang, F., Govig, J., Kirsch, A., Chan, K., Lai, K., Wu, S., Dhoot, S.G., Kumar, A.R., Agiwal, A., Bhansali, S., Hong, M., Cameron, J., et al.: Mesa: geo-replicated, near real-time, scalable data warehousing. PVLDB **7**, 1259–1270 (2014)
36. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google file system. SIGOPS Oper. Syst. Rev. **37**, 29–43 (2003)
37. Fay, C., Jeffrey, D., Sanjay, G., et al.: Bigtable: a distributed storage system for structured data. ACM Trans. Comput. Syst. **26**, 4:1–4:26 (2008)
38. Lamport, L.: Paxos made simple. ACM SIGACT News (Distrib. Comput. Column) **32**, 51–58 (2001)
39. Stonebraker, M., Abadi, D., DeWitt, D.J., Madden, S., Paulson, E., Pavlo, A., Rasin, A.: MapReduce and parallel DBMSs: friends or foes? Commun. ACM **53**, 64–71 (2010)
40. Hall, A., Bachmann, O., Büssow, R., Gănceanu, S., Nunkesser, M.: Processing a trillion cells per mouse click. VLDB Endow. **5**, 1436–1446 (2012)
41. Xu, Y., Kostamaa, P., Gao, L.: Integrating hadoop and parallel DBMs. In: Proceedings of SIGMOD International Conference on Management of Data, pp. 969–974 (2010)
42. Friedman, E., Pawlowski, P., Cieslewicz, J.: SQL/MapReduce: a practical approach to self-describing, polymorphic, and parallelizable user-defined functions. VLDB Endow. **2**, 1402–1413 (2009)
43. Melnik, S., Gubarev, A., Long, J.J., Romer, G., Shivakumar, S., Tolton, M., Vassilakis, T.: Dremel: interactive analysis of web-scale datasets. Commun. ACM **54**, 114–123 (2011)
44. DeWitt, D.J., Halverson, A., Nehme, R., Shankar, S., Aguilar-Saborit, J., Avanes, A., Flasza, M., Gramling, J.: Split query processing in polybase. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 1255–1266 (2013)
45. Pedro, E., Rocha, P., Luis, E.d.B., Chris, C.: Cubrick: a scalable distributed MOLAP database for fast analytics. In: Proceedings of International Conference on Very Large Databases, pp. 1–4 (2015)
46. Gupta, A., Agarwal, D., Tan, D., Kulesza, J., Pathak, R., Stefani, S., Srinivasan, V.: Amazon redshift and the case for simpler data warehouses. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 1917–1923 (2015)
47. Yang, F., Tschetter, E., Léauté, X., Ray, N., et al.: Druid: a real-time analytical data store. In: Proceedings of ACM International Conference on Management of Data, pp. 157–168 (2014)
48. Lamb, A., Fuller, M., Varadarajan, R., Tran, N., Vandiver, B., Doshi, L., Bear, C.: The vertica analytic database: C-store 7 years later. VLDB Endow. **5**, 1790–1801 (2012)

49. Valiant, L.G.: A bridging model for parallel computation. Commun. ACM **33**, 103–111 (1990)
50. Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., Hellerstein, J.M.: Distributed GraphLab: a framework for machine learning and data mining in the cloud. Proc. VLDB Endow. **5**, 716–727 (2012)
51. Simmhan, Y., Wickramaarachchi, C., Kumbhare, A.G., Frîncu, M., Nagarkar, S., Ravi, S., Raghavendra, C.S., Prasanna, V.K.: Scalable analytics over distributed time-series graphs using goffish. CoRR abs/1406.5975 (2014)
52. Shao, B., Wang, H., Li, Y.: Trinity: a distributed graph engine on a memory cloud. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 505–516 (2013)
53. Mayer, R., Mayer, C., Tariq, M.A., Rothermel, K.: GraphCEP: real-time data analytics using parallel complex event and graph processing. In: Proceedings of the ACM International Conference on Distributed and Event-based Systems, pp. 309–316 (2016)
54. Mayer, R., Koldehofe, B., Rothermel, K.: Predictable low-latency event detection with parallel complex event processing. IEEE Internet Things J. **2**, 1 (2015)
55. Acharjya, D.P., Ahmed, K.: A survey on big data analytics: challenges, open research issues and tools. Int. J. Adv. Comput. Sci. Appl. **7**, 511–518 (2016)
56. Inoubli, W., Aridhi, S., Mezni, H., Jung, A.: An experimental survey on big data frameworks. ArXiv e-prints, pp. 1–41 (2017)
57. Madhuri, T., Sowjanya, P.: Microsoft Azure v/s Amazon AWS cloud services: a comparative study. J. Innov. Res. Sci. Eng. Technol. **5**, 3904–3908 (2016)
58. Pkknen, P., Pakkala, D.: Reference architecture and classification of technologies, products and services for big data systems. Big Data Res. **2**, 166–186 (2015)
59. Landset, S., Khoshgoftaar, T.M., Richter, A.N., Hasanin, T.: A survey of open source tools for machine learning with big data in the hadoop ecosystem. J. Big Data **2**, 1–36 (2015)
60. Khalifa, S., Elshater, Y., Sundaravarathan, K., Bhat, A., Martin, P., Imam, F., Rope, D., et al.: The six pillars for building big data analytics ecosystems. ACM Comput. Surv. **49**, 33:1–33:36 (2016)
61. Poleto, T., de Carvalho, V.D.H., Costa, A.P.C.S.: The roles of big data in the decision-support process: an empirical investigation. In: Delibašić, B., Hernández, J.E., Papathanasiou, J., Dargam, F., Zaraté, P., Ribeiro, R., Liu, S., Linden, I. (eds.) ICDSST 2015. LNBIP, vol. 216, pp. 10–21. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18533-0_2
62. Lahcene, B., Ladjel, B., Yassine, O.: Coupling multi-criteria decision making and ontologies for recommending DBMS. In: Proceedings of International Conference on Management of Data (2017)
63. Sahri, S., Moussa, R., Long, D.D.E., Benbernou, S.: DBaaS-expert: a recommender for the selection of the right cloud database. In: Andreasen, T., Christiansen, H., Cubero, J.-C., Raś, Z.W. (eds.) ISMIS 2014. LNCS (LNAI), vol. 8502, pp. 315–324. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08326-1_32

Towards a Digital Business Operating System

Jan Bosch(✉)

Chalmers University of Technology, Gothenburg, Sweden
jan@janbosch.com

Abstract. With the increasingly important role of software in industry and society overall, the traditional ways of organizing are becoming increasingly outdated. To remain competitive, companies need to adopt a new, digital business operating mechanism. In this paper, we present such a system consisting of four dimensions, i.e. speed, data, ecosystems and empowerment, and three scopes of work, i.e. operations, development and innovation.

Keywords: Agile practices · Data-driven development · Software ecosystems Empowerment

1 Introduction

Software is eating the world, Marc Andreessen wrote in his Wallstreet Journal OpEd [1]. Industry after industry is seeing a fundamental shift in R&D investment away from mechanics and electronics and towards software is increasing [5]. This is driven by the fact that it is the software in modern products, rather than the mechanics and hardware, defines the value. Often referred to as digitalization, this transformation plays an increasingly important role in the industry and it has profound implications on the way software-intensive systems companies operate.

In this article, we analyze these implications in more detail. To do so, we first analyze the key trends in industry and society that we have identified, ranging from the transition from products to services to the constantly growing size of software in typical systems. Based on these trends, we identify four key factors, i.e. speed, data, ecosystems and empowerment. We apply these four key factors to three scopes of activity, i.e. operations, development and innovation. Based on our collaboration with industry, we suggest that this model is critical for the software-intensive systems industry to adopt going forward as companies are under severe pressure to improve their capability to deliver on these software-driven needs.

The work presented in this article has been conducted in the context of Software Center (www.software-center.se), a collaboration around software engineering research between more than 10 companies, including Ericsson, Volvo Cars, Grundfos, Saab, Jeppesen (part of Boeing), Siemens and Bosch, and five Swedish universities. The findings presented here are consequently based on significant amounts of industry experience.

The remainder of the paper is organized as follows. The next section introduces what we see as the key trends affecting the industry. Subsequently, we present an overview of the new digital business operating system that companies need to adopt in

E. Cabello et al. (Eds.): ICSOFT 2017, CCIS 868, pp. 296–308, 2018.
https://doi.org/10.1007/978-3-319-93641-3_14

order to survive and thrive in the digital transformation. The following sections then describe in some more detail the aforementioned four key factors. Finally, we discuss the typical types of activities that exist in the organization, i.e. operations, development and innovation, and describe how these relate to the key factors. Finally, we conclude the paper by summarizing our findings.

2 Trends in Industry and Society

To an attentive observer, it is clear that the world is changing continuously and that the pace of this change is accelerating. This change is driven by the constant evolution of technology and especially by digital technology. Digital technologies experience exponential improvements due to Moore's Law concerning chips and microprocessors (the doubling of transistor density every 18 months). However, there are other technologies in play as well. For instance, the cost of transporting a bit halves every nine months and the cost of storing a bit in digital storage halves every twelve months. These exponential technology developments lead to fundamental shifts in industry and in society and give cause to several trends. In this section, we discuss some of these trends in more detail.

2.1 Shifting Nature of Product Innovation

Especially in the embedded systems industry, for a long time the key technology that received attention for innovation were the mechanical parts of the system or product. Even if the product contains electronics and software, these technologies were treated as secondary and not necessarily central to the product. This is clearly illustrated by the development process for software as this was subjugated to the process for mechanical systems.

The key trend is that software has become the differentiating technology for many products whereas mechanics and hardware (electronics) are rapidly becoming commodity. System architecture separates the mechanics and electronics from the software, resulting two, largely independent release processes. This allows software to be updated very frequently, both before the product leaves the factory and after it has been deployed in the field at customers.

In the Software Center, several companies are undergoing this transformation. For instance, AB Volvo estimates that 70% of all innovation in their trucks is driven by software. Volvo Cars estimates that 80–90% of their innovation is driven by electronics and software. Over the last decade, the R&D budget at telecom company Ericsson has shifted towards software, which now represents more than 80% of the budget.

2.2 From Products to Services

Both in the B2C and in the B2B world, there is a shift taking place from ownership to access and from CAPEX to OPEX. One of the reasons is that it allows companies to rapidly change course when customer demand changes. Especially newer generations such as Generation Y and the millennials have changed their values from owning to having access to expensive items[1]. For instance, the typical car is used less than an hour per day and is not used the other 23+ h.

Because of this development, many companies transition from selling products to delivering services. This requires significant changes to business models but also means that the products now become a cost rather than revenue generators, changing the key incentives for companies. For instance, maximizing the economic life of the product after deployment is an important cost reduction measure, which often requires deploying new software in products in the field.

To illustrate this point, the fastest growing business unit of Ericsson is its global services business. This unit has grown faster in terms of revenue and staff than the product units. Also, automotive companies expect that by the mid 2020, more than half of their cars will be utilized through service agreements rather than through ownership.

2.3 From Technology- to Customer-Driven Innovation

Although technology forms the foundation for innovation, for several industries, despite the use of patents and other IP protection mechanisms, new technologies tend to become available to all players at roughly the same time. This causes these companies to have very little benefit in terms of differentiation because of new technologies. In response, companies increasingly prioritize customer-driven innovation [3]. This requires deep engagement with customers as well as quantitatively analyzing customer behavior using collected data from instrumentation of deployed software systems, both online and offline. In [3], we study several case companies that adopted new techniques to collect more customer insight as part of their product development.

2.4 The Size of Software

Depending on the industry, the size of software in software-intensive systems increases with an order of magnitude every five to ten years. The main challenge is that a software system that is 10 times the size of a previous generation requires new architectural approaches, different ways of organizing development, significant modularization of testing, release and post-deployment upgrades as well as the complications of running a larger R&D organization.

Although there are several studies documenting this trend, one of the most illustrative studies is by Ebert and Jones [5] that analyze this trend for embedded systems

2.5 Need for Speed

As discussed in the introduction, in society, we see a continuous acceleration of user adoption of new technologies, products and solutions. For example, it took Facebook 10 months to reach a million users whereas the mobile app "draw something" reached that number in just days. With the "consumerization" of the enterprise, also inside corporations the adoption of new applications, technologies and systems accelerates.

[1] https://en.wikipedia.org/wiki/Sharing_economy.

Companies need to respond to new customer needs and requests at unprecedented speeds. This requires a level of enterprise-wide agility that is often exceedingly difficult to accomplish in conventional, hierarchical organizations. This “need for speed” requires different ways of organizing, building and architecting software and software development in companies.

2.6 Playing Nice with Others

Many software-intensive systems industries become increasingly aware of the opportunities that exist when using one’s ecosystem of partners in a more proactive and intentional fashion. The strategic challenge has shifted from internal scale, efficiency and quality and serving customers in a one-to-one relationship to creating and contributing to an ecosystem of players, including, among others, suppliers, complementors, customers and potentially even competitors. The ecosystem trend is not just visible in the mobile industry with its app stores, but also in B2B markets such as those surrounding SAP and Microsoft Office. Establishing and evolving an ecosystem of partners of different types has become the key differentiator in several industries and may easily be the cause of winning in a market or being relegated to a less dominant position. In [4] we discuss several cases that show improved competitiveness of companies that effectively use their ecosystem.

3 Towards a Digital Business Operating System (DiBOS)

The trends that we discussed in the previous section have one important commonality: the traditional way of building and deploying products using waterfall-style approaches, requirements-driven prioritization and standardization of work processes is falling short. The traditional company is unable to meet these trends in a fashion that allows it to stay competitive. The problem is that it is not one change that is required or one function that needs to transform. Instead, it is the entire business operating system that needs to be replaced. In order to succeed in a digitizing world, organizations need to adopt a new digital business operating system (DiBOS) [2].

As shown in Fig. 1, DiBOS consists of four dimensions and three scopes of work. The four dimensions are speed, data, ecosystems and empowerment. The scopes of work include operations, development and innovation. In the rest of this section, we describe the four dimensions. In the next section, we describe the scopes of work.

3.1 Speed

The ability of companies to rapidly convert identified customer needs into working solutions in the hands of customers is increasingly important for maintaining a competitive position. The times where companies could spend years developing new products is long gone and customers expect fast responses to their needs and continuous improvement of the functionality in their systems.

Our research over the last decade has shown that companies evolve in predictable ways. The main source of variation is timing, i.e. when changes happen, not the changes that actually are driven forward. In Fig. 2, we show the typical evolution path for companies.

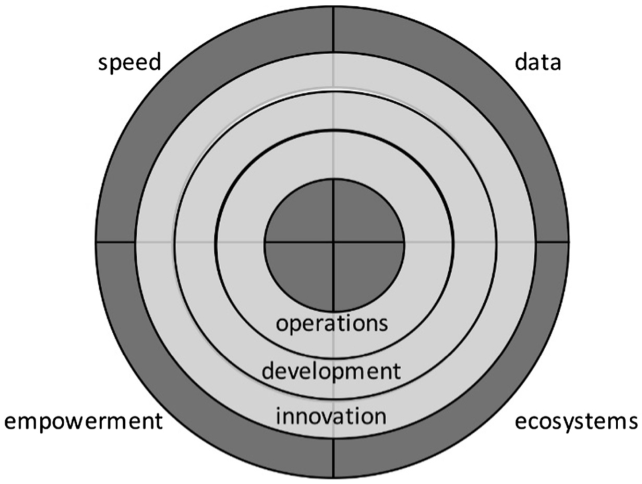


Fig. 1. Illustrating DiBOS.

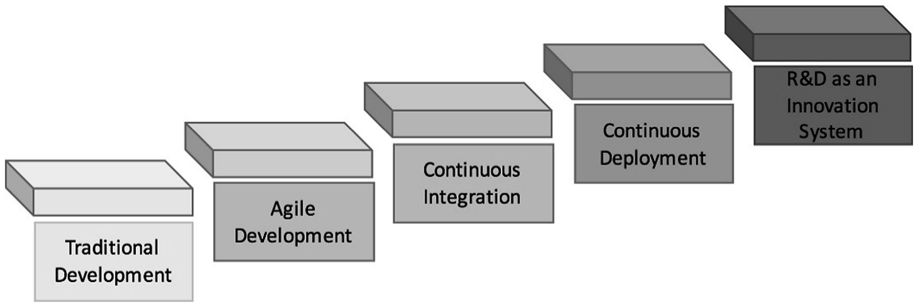


Fig. 2. Speed dimension of DiBOS.

As the figure shows, companies evolve through five stages:

- **Traditional Development.** This indicates the starting point for companies before any modern development methods are adopted. The traditional exhibits many aspects of a waterfall style approach. These aspects include a relatively long time between the decision of what to build and the delivery of the associated functionality. Also, typically there is a sequential process where requirements, architecture design, detailed design, implementation, testing and release are performed sequentially. Finally, the R&D organization is functionally organized based on the process steps.

- **Agile Development.** One of the inefficiencies of traditional approaches is the number of internal handovers between different functions. In response to these and other inefficiencies, many organizations have adopted several agile practices, including teams, sprints, backlogs, daily standups, clear definitions of done, etc. Adoption of agile practices focuses predominantly on the agile team itself and to a lesser extent on the interactions between the team and the rest of the organization.
- **Continuous Integration.** Once the agile teams have been established and are operating based on agile principles, the attention shifts towards making sure that these teams are building code that actually works. Continuous integration, typically a set of software systems that build and test software immediately after it is checked in, offers as the main advantage that the organization has constant and accurate insight into the state of development.
- **Continuous Deployment.** When continuous integration is established and institutionalized, there always is a production quality software version available. Once customers realize this, they will demand early access to the software assuming the new features provide benefit for them. This pressure from the market often leads to continuous deployment, i.e. the frequent (at least once per agile sprint) release of new software to some or all customers.
- **R&D as an Experiment System.** Once continuous deployment is established for some or all customers, organizations realize that one can also test the implications of new features on customers and systems in the field by deploying partially implemented features to verify that the expected value of a new feature is indeed realized. This allows companies to redirect or stop development of features if the data from the field are not in line with expectations. This allows for a significant increase in the accuracy of R&D investments.

3.2 Data

Data is often referred to as the new oil. Every company is eager to collect and store data, but the effective use of that data is not always as well established as one would wish. Also, the type of data collected by companies traditionally not concerned with data tends to be concerned with quality issues such as defects, troubleshooting and logging. Research by others and us shows, for instance, that more than half of all features in a typical software system are hardly ever or never used. However, because companies frequently fail to collect data about feature usage. There is no data that can be used for building the right features or removing features that are not used.

Our research shows that companies evolve through their use of data in a repeatable and predictable pattern.

The evolution pattern is shown in Fig. 3 and below we describe each step in some more detail:

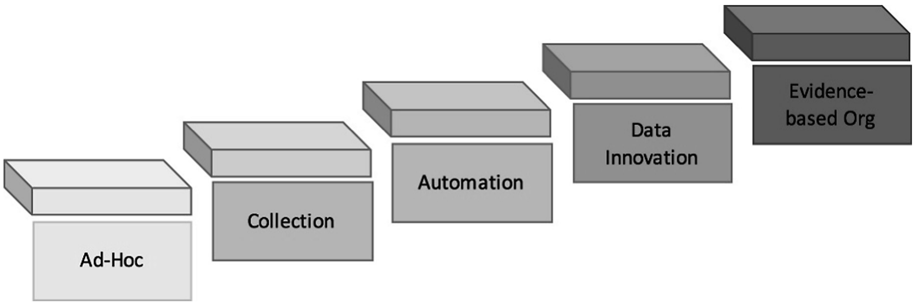


Fig. 3. Data dimension of DiBOS.

- **Ad-Hoc.** At the lowest level, the organization has no systematic use of data concerning the performance of the system or the behavior of users. Not only are data not used, they are not even collected or analyzed. In the cases someone in the organization decides to use data, the collection, analysis, reporting and decision making based on the data all need to be done manually.
- **Collection.** Once a certain level of awareness of the relevance of data is established in the organization, the next step is to start to instrument the software in its products and systems with the intent of putting the data in a data warehouse. Although initially the goal will be to collect as much data as possible, soon the discussion turns to what types of data should be collected and for what purposes.
- **Automation.** As managers and others provide requests to the data analytics team, it will become clear that certain queries come back frequently. When this happens, often dashboards are defined and initiated. At this stage, the collection, analysis and reporting on the data are automated and decision making is supported.
- **Data Innovation.** As dashboards and continuous reporting of data become the norm, the awareness of the limitations of static dashboards becomes apparent. In response, data analysts and users of the dashboards start to collaborate to drive a continuous flow of new metrics and KPIs, typically replacing existing ones, resulting in a dynamically evolving data-driven way of working.
- **Evidence-based Organization.** Once we reach the final stage, the entire organization, and not only R&D, uses data-driven decision making and experimentation considered to be the most powerful tool to accomplish this. The organization has adopted Edwards Deming's motto: In God we trust; all others must bring data.

3.3 Ecosystems

Upon hearing the word "ecosystem", most in the software industry think of app stores for mobile phones. However, any company is part of multiple ecosystems [6], including an innovation ecosystem to share the cost and risk of innovation, a differentiation ecosystem where the company looks to complement its own solutions with functionality offered by others to strengthen the value proposition to its customers and a commodity functionality ecosystem where the company seeks to limit its own R&D resource investment by engaging in the ecosystem to share maintenance cost, replace internal software with COTS or OSS components, etc (Fig. 4).

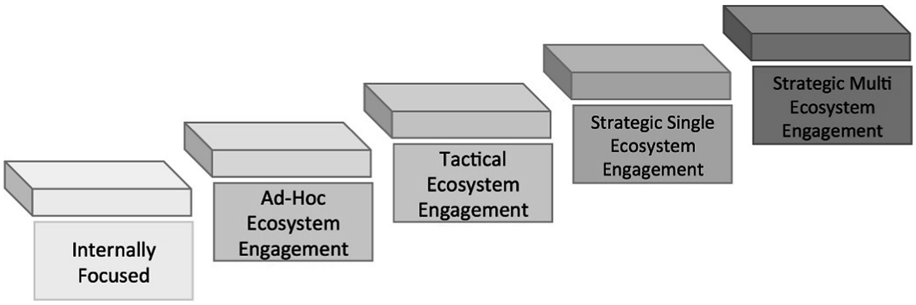

Internally Focused
Ad-Hoc Ecosystem Engagement
Tactical Ecosystem Engagement
Strategic Single Ecosystem Engagement
Strategic Multi Ecosystem Engagement

Fig. 4. Ecosystem dimension of DiBOS.

Similar to the other dimensions, we have identified that companies engage with their ecosystems in a predictable pattern, evolving from being internally focused to strategically engaging with multiple ecosystems. Below we discuss each step in some more detail:

- **Internally Focused.** The first and basic stage is where the company is exclusively internally focused. As no company is an island, the company, of course, has to interact with other companies, but the basic principle is that everything that can be done in house is done internally.
- **Ad-hoc Ecosystem Engagement.** The next step is where the company starts to engage the ecosystem in an ad hoc fashion. Often driven by some crisis or cost efficiency program, the company realizes that some issue or problem could be addressed by stopping to do something by itself and to give it to another organization.
- **Tactical Ecosystem Engagement.** Once the first step towards relying on an outside company has been accomplished and the results are satisfactory, there is a shift in the culture where more areas that are contextual for a company are considered for outsourcing. The initial approach tends to be more tactical in nature, meaning that the selection of partners and collaboration with these partners is more structured than in the previous step, but still the engagement is tactical in nature.
- **Strategic Single Ecosystem Engagement.** The next step is where the company starts to build long-term relationships where co-evolution in the context of a symbiotic relationship can be realized. At this stage, the companies also collaborate outside of individual contracts and perform joint strategy development, transition responsibility for certain types of functionality to each other and together look for ways to increase the overall value in the ecosystem.
- **Strategic Multi Ecosystem Engagement.** The final stage is where the company has matured to the point that it can handle all its ecosystems in a strategic fashion. Here, the company allocates its own resources predominantly to differentiation and relies on and orchestrates its ecosystems for collaborative innovation and sharing the cost of commodity functionality.

3.4 Empowerment

One of the surprising, but often ignored, facts of life is that organizations almost all look the same. Starting from a CEO, there is a group of functional leaders in the C-suite and below that there are multiple levels of managers until we reach front-line people that actually do some work. This traditional, hierarchical way of organizing has served the western world well for most of the 20th century, but as the business environment experiences an increasingly rapid pace of change, it becomes harder and harder for these traditional organizations to stay competitive. Over the last decade, a new class of organizations has emerged that is concerned with empowerment of individuals, removes the formal manager role so that people do not have a boss and introducing alternative mechanisms for coordinating between individuals in the organization.

In Fig. 5, we show the stages that an organization evolves through to transition from a hierarchical organization to an empowered organization. Below, we describe these stages in some more detail:

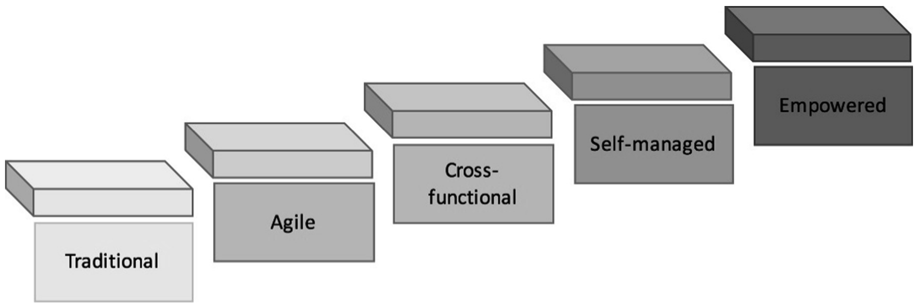


Fig. 5. Empowerment dimension of DiBOS.

- **Traditional.** The traditional organization is the hierarchical organization where employees report to managers who report to manager until the CEO is reached. In these companies, the culture, ways of working, formal operating mechanisms, etc. are all driven by hierarchy, communication up and down the line, etc.
- **Agile.** The first step towards toward empowerment is provided by adopting agile practices. When going agile, at least teams receive a higher degree of autonomy and have more freedom on how they work, even if decisions related to what they should work on are still driven by the traditional organization.
- **Cross-functional.** When adopting agile practices beyond individual teams, often there is a need to involve functions upstream, such as product management, and downstream, such as the release organization. This often leads to cross-functional ways of working as well as cross-functional teams that have high degrees of autonomy. At this stage, management shifts toward outcomes and cross-functional organizations are left to their own devices as long as they deliver on the desired outcomes.

- **Self-managed.** When the end-to-end R&D organization has successfully adopted empowerment, the rest of organization is likely to follow suit, resulting in an organization where every individual and team is self-managed. The challenge at this stage is that even though everyone is self-managing, the culture often still expects or relies on some hierarchical operating mechanisms such as a setting business strategy.
- **Empowered.** In the final stage, every part of the organization is fully empowered and operates based on peer-to-peer coordination, evolving strategies and frequent self-reflection on performance and the quality of the organization for its employees.

4 Operationalizing DiBOS

In the previous section, we discussed the four dimensions of DiBOS. However, there is a second dimension to how companies function. As we showed in Fig. 1, there are three types of activity, i.e. operations, development and innovation. In the next sections, we discuss each type and its relation to the four dimensions discussed earlier.

4.1 Operations

Once a company puts a product or service in market, an operations organization will be required that addresses the lifecycle of product or service. This starts from marketing and sales, followed by installation and start of operation at the customer, ongoing support, releases of updates, maintenance and finally sun-setting. Depending on the product or service this lifecycle may take decades, as is the case for military equipment and telecommunications infrastructure, or months, as is the case for many mobile phones.

Although the four dimensions of DiBOS were developed primarily for development, these are relevant for operations as well:

- **Speed.** During operations, agility and responsiveness clearly is a differentiator for companies as it provides customers with a rapid response to their issues and concerns. Also, through the use of continuous deployment, the company can identify and resolve issues before customers even realize that there is a problem.
- **Data.** In the case of connected products and online services, insight into customer behavior and the performance of deployed systems is incredibly powerful as a mechanism to proactively serve customers. In addition, it can be very effectively used for cross-selling and up-selling of services to customers as the company has insight into the operational performance of its customers.
- **Ecosystems.** Similar to development, also the operations organization faces decisions on what to do by itself and were to orchestrate its ecosystem and rely on partners for certain aspects of operations.
- **Empowerment.** Finally, especially in operations organizations where many of the staff are no white-collar professionals, there is a tendency to standardize processes and to force people in a rigid operational model. However, empowering these

people when having established the right culture often is incredibly powerful in terms of the quality of service offered to customers while finding cost-effective solutions for the company itself.

4.2 Development

The four dimensions of DiBOS were initially developed for the development and R&D organization of the company. Our experience that R&D in many ways is the embodiment of the business strategy as this part of the company takes all the design decisions that enable or complicate certain business strategies. As the business and operational organizations often become aware of the need for specific solutions, R&D either has prepared for these needs much earlier or will need to scramble to respond to the needs in the market.

As DiBOS has initially been developed for development predominantly, the four dimensions relate to this activity as follows:

- **Speed.** The speed dimension is concerned with agile practices, continuous integration of functionality and continuous deployment at customers. Climbing the speed dimension allows the company to shorten many of its feedback cycles and transition from opinion-based decision making to data-driven decision making.
- **Data.** Our research shows that half or more of the features developed for typical software systems are never used or used so seldom that the feature does not provide a relevant return on investment on the R&D cost in terms of business value. Instrumenting software that is frequently deployed allows for iterative development of features to ensure, with each iteration, that the R&D investment is resulting in the desired outcome.
- **Ecosystems.** Our TeLESM model [6] suggests that a product or family of related products are part of three ecosystems, i.e. an innovation, a differentiation and a commodity ecosystem. Each of these ecosystems needs to be engaged with different drivers and success metrics and the ecosystem dimension describes how to accomplish this.
- **Empowerment.** Traditional development approaches assumed a requirement centric way of working, but in our experience organizations that are mature in their strategy development and are able to convert this strategy into quantitative outcome metrics can shift its teams to outcome driven development. This means that teams do not build towards specifications, but rather experiment with the aim of accomplishing a certain outcome. This is allows for more empowerment of teams.

4.3 Innovation

Innovation is the lifeblood of any organization and those that fail on this lose their competitive position over time as more and more of their offerings to customers become commoditized. In many ways, innovation is the predecessor of development as it seeks to find those items that are worthwhile to customers and that should be developed in a mature, productive fashion [3].

DiBOS offers several tools and enables for innovation in the four dimensions as we discuss below:

- **Speed.** Innovation aims at testing as many ideas as possible with customers and other ecosystem partners against the lowest cost per experiment. Speed in development and deployment of these experiments with customers is a key enabler for innovation.
- **Data.** One of the key dangers of innovation is to rely on the opinions of senior leaders in the organization, or the opinions of anyone else for that matter. Instead, prioritization of innovation efforts should be based on data about customer behavior as well as the behavior of systems deployed in the field.
- **Ecosystems.** The TeLESM model [6] that we mentioned in the previous section explicitly identifies the innovation layer and ecosystem as a key part of the model. Sharing the cost and risk of innovation is an important factor for many companies as it allows for a wider innovation funnel.
- **Empowerment.** One of the painful lessons for many organizations is that the more senior the leader in the organization, the more likely it is that the opinion of this leader concerning an innovative idea is wrong. Instead, we need to allow individuals and teams to use a part of their time to pursue their innovative ideas without having to worry about management interference. This obviously requires empowerment of individuals and teams.

5 Conclusion

As Marc Andreessen wrote in his Wallstreet Journal OpEd [1], software is eating the world. Across the industry we see a fundamental shift in R&D investment from “atoms”, i.e. mechanics and electronics, to “bits”, i.e. software [5]. In modern products, value is shifting to the software, rather than the mechanics and hardware. Often referred to as digitalization, this transformation plays a disrupting role in the industry and it has profound implications on the way software-intensive systems companies operate.

In this paper, we analyzed the key trends in industry and society that are causing this disruption, ranging from the transition from products to services to the constantly growing size of software in typical systems. Our conclusion is that companies need to adopt a new digital business operating system (DiBOS). DiBOS consists of four dimensions, i.e. speed, data, ecosystems and empowerment, that are central for the software-intensive systems industry going forward as companies are under severe pressure to improve their capability to deliver on these software-driven needs. In addition, it considers three scopes of work, i.e. operations, development and innovation. Our research shows that DiBOS and its constituent parts offer a comprehensive model to evolve towards in order to become a digital company.

The work presented in this article has been conducted in the context of Software Center (www.software-center.se), a collaboration around software engineering research between eleven companies, including Ericsson, Volvo Cars, Grundfos, Saab, Jeppesen (part of Boeing), Siemens and Bosch, and five Swedish universities. The findings presented here are consequently based on significant amounts of industry experience.

In future work, we aim to further develop the methods, techniques and tools that underlie DiBOS as well as continue to validate the model in more industry contexts in order to ensure its relevance and applicability.

References

1. Andreessen, M.: Why software is eating the world. Wall Str. J. 20 August 2011. http://www.wsj.com/articles/SB10001424053111903480904576512250915629460. Accessed 18 Feb 2015
2. Bosch, J.: Speed, Data and Ecosystems: Excelling in a Software-Driven World. CRC Press, Boca Raton (2017)
3. Bosch-Sijtsema, P., Bosch, J.: User involvement throughout the innovation process in high-tech industries. J. Prod. Innov. Manag. **32**(5), 793–807 (2014)
4. Bosch-Sijtsema, P.M., Bosch, J.: Plays nice with others? Multiple ecosystems, various roles and divergent engagement models. Technol. Anal. Strat. Manag. **27**, 960–974 (2015)
5. Ebert, C., Jones, C.: Embedded software: facts, figures, and future. IEEE Comput. **42**, 42–53 (2009)
6. Olsson, H.H., Bosch, J.: From Ad-Hoc towards strategic ecosystem management: the three-layer ecosystem strategy model. J. Softw. Evol. Process **29**(7), e1876 (2017). https://doi.org/10.1002/smr.1876

Author Index