Joaquim Filipe
Leszek A. Maciaszek (Eds.)

Communications in Computer and Information Science 417

Evaluation of Novel Approaches to Software Engineering

8th International Conference, ENASE 2013
Angers, France, July 2013
Revised Selected Papers

Springer

Communications in Computer and Information Science 417

Joaquim Filipe Leszek A. Maciaszek (Eds.)

Evaluation of Novel Approaches to Software Engineering

8th International Conference, ENASE 2013
Angers, France, July 4-6, 2013
Revised Selected Papers

Springer

Volume Editors

Joaquim Filipe
INSTICC and IPS, Estefanilha, Setúbal, Portugal
E-mail: joaquim.filipe@estsetubal.ips.pt

Leszek A. Maciaszek
Wrocław University of Economics, Poland
and
Macquarie University, Sydney, NSW, Australia
E-mail: leszek.maciaszek@mq.edu.au

ISSN 1865-0929 e-ISSN 1865-0937
ISBN 978-3-642-54091-2 e-ISBN 978-3-642-54092-9
DOI 10.1007/978-3-642-54092-9
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013957792

CR Subject Classification (1998): D.2, F.3, D.3, H.4, K.6

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This Springer volume contains the papers of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE) held in Angers, France, sponsored by the Institute for Systems and Technologies of Information, Control and Communication (INSTICC) and supported by Oracle, Conseil général de Maine–et–Loire, Conseil Régional des Pays de la Loire, Ville Angers and Angers Loire Tourisme.

The conference and the papers in this Communications in Computer and Information Science (CCIS) series volume reflect a growing effort to increase the dissemination of new results among researchers and professionals related to the evaluation of novel approaches to software engineering. The ENASE 2013 conference built on successes of previous conferences that took place as follows:

- 2006 – Erfurt, Germany
- 2007 – Barcelona, Spain
- 2008 – Madeira, Portugal
- 2009 – Milan, Italy
- 2010 – Athens, Greece
- 2011 – Beijing, China
- 2012 – Wrocław, Poland
- 2013 – Angers, France

By comparing novel approaches with established traditional practices and by evaluating them against software quality criteria, the ENASE conferences advance knowledge and research in software engineering, identify the most hopeful trends, and propose new directions for consideration by researchers and practitioners involved in large-scale software development and integration.

ENASE 2013 received 46 submissions, of which 24% were presented as full papers. Additionally, 28% were short oral presentations and 15% were presented as posters. To evaluate each submission, a double-blind paper evaluation method was used: Each paper was reviewed by at least two experts from the International Program Committee in a double-blind review process, and most papers had three reviews or more.

The best full papers of the conference were invited, after corrections and extensions, to appear in this CCIS book. The book is a continuation of the previous ENASE volumes that have appeared in the CCIS series. All ENASE conferences, except the first one in 2006, had such Springer publications.

ENASE 2013 was held in conjunction with the 15th International Conference on Enterprise Information Systems (ICEIS 2013). The two conferences shared the same list of keynote speakers in joint plenary sessions. The prominent list of 2013 keynotes consisted of the following experts:

- Stephen Mellor, Freeter, UK
- Fabien Gandon, Inria, France
- Ulrich Frank, University of Duisburg-Essen, Germany
- Henderik A. Proper, Public Research Centre - Henri Tudor, Luxembourg

December 2013

Joaquim Filipe
Leszek Maciaszek

Organization

Conference Chair

Joaquim Filipe	Polytechnic Institute of Setúbal/INSTICC, Portugal

Program Chair

Leszek Maciaszek	Wroclaw University of Economics, Poland/ Macquarie University, Sydney, Australia

Organizing Committee

Marina Carvalho	INSTICC, Portugal
Helder Coelhas	INSTICC, Portugal
Bruno Encarnação	INSTICC, Portugal
Ana Guerreiro	INSTICC, Portugal
André Lista	INSTICC, Portugal
Andreia Moita	INSTICC, Portugal
Carla Mota	INSTICC, Portugal
Raquel Pedrosa	INSTICC, Portugal
Vitor Pedrosa	INSTICC, Portugal
Ana Ramalho	INSTICC, Portugal
Susana Ribeiro	INSTICC, Portugal
Sara Santiago	INSTICC, Portugal
Mara Silva	INSTICC, Portugal
José Varela	INSTICC, Portugal
Pedro Varela	INSTICC, Portugal

Program Committee

Guglielmo de Angelis, Italy
Maria Bielikova, Slovak Republic
Piotr Bubacz, Poland
Dumitru Burdescu, Romania
Wojciech Cellary, Poland
Rebeca Cortazar, Spain
Massimo Cossentino, Italy
Bernard Coulette, France
Marcelo d'Amorim, Brazil
Philippe Dugerdil, Switzerland
Angelina Espinoza, Spain
Joerg Evermann, Canada
Maria João Ferreira, Portugal
Agata Filipowska, Poland
Maria Ganzha, Poland
Juan Garbajosa, Spain

Cesar Gonzalez-Perez, Spain
Rene Hexel, Australia
Benjamin Hirsch, UAE
Charlotte Hug, France
Bernhard G. Humm, Germany
Zbigniew Huzar, Poland
Akira Imada, Belarus
Stefan Jablonski, Germany
Slinger Jansen, The Netherlands
Monika Kaczmarek, Poland
Robert S. Laramee, UK
George Lepouras, Greece
Pericles Loucopoulos, UK
Graham Low, Australia
Jian Lu, China
André Ludwig, Germany
Ivan Lukovic, Serbia
Leszek Maciaszek, Poland and Australia
Lech Madeyski, Poland
Leonardo Mariani, Italy
Sascha Mueller-Feuerstein, Germany
Johannes Müller, Germany
Andrzej Niesler, Poland
Janis Osis, Latvia
Mourad Oussalah, France
Marcin Paprzycki, Poland
Dana Petcu, Romania
Naveen Prakash, India
Elke Pulvermueller, Germany
Rick Rabiser, Austria
Lukasz Radlinski, Poland
Artur Rot, Poland
Radoslaw Rudek, Poland
Francisco Ruiz, Spain
Krzysztof Sacha, Poland
Motoshi Saeki, Japan
Jakub Swacha, Poland
Stephanie Teufel, Switzerland
Rainer Unland, Germany
Olegas Vasilecas, Lithuania
Krzysztof Wecel, Poland
Michael Whalen, USA
Igor Wojnicki, Poland
Kang Zhang, USA

Auxiliary Reviewers

Jaap Kabbedijk, The Netherlands
Salvatore Lopes, Italy
Tomás Martínez-Ruiz, Spain
Marian Cristian Mihaescu, Romania
Djordje Obradovic, Serbia
Patrizia Ribino, Italy

Invited Speakers

Stephen Mellor — Freeter, UK
Fabien Gandon — Inria, France
Ulrich Frank — University of Duisburg-Essen, Germany
Henderik A. Proper — Public Research Centre - Henri Tudor, Luxembourg

Table of Contents

Designing a Virtual Reality Software: What Is the Real Contribution of End-Users to the Requirements Prioritization?

Emilie Loup-Escande[1] and Olivier Christmann[2]

[1] Centre de Recherches en Psychologie, Cognition et Communication, Université Rennes 2, Place du recteur Henri Le Moal, 35043 Rennes Cedex, France
emilie.loup-escande@uhb.fr
[2] Arts et Métiers ParisTech, LAMPA, 2 Boulevard du Ronceray, 49000 Angers, France
olivier.christmann@ensam.eu

Abstract. This paper deals with the requirements prioritization which is a step, or an activity, of design process influencing the decisions for the development of software products. This step is often implemented by designers and uncommonly by users. The objective of this paper is consequently to characterize the implementation of requirements prioritization by end-users. For that, we examined literatures of requirements engineering and ergonomics and we conducted an empirical study with twenty end-users of a virtual reality software. In this study, we analyze the lists of prioritized functionalities and the functionalities evoked spontaneously by users. Results show that (1) the priority functionalities for users were not systematically implemented by designers, (2) the different priority levels depended on users' profiles, (3) the users who assigned 'important' and 'unimportant' priority levels evoked additional functionalities, and (4) the spontaneously evoked functionalities were mainly precisions of anticipated functionalities.

Keywords: Prioritization, Requirement, Virtual Reality, Software Design Process.

1 Introduction

Requirements prioritization is a crucial activity in software design. Numerous research works deal with the study and the improvement of this part of the software process, for example in the field of requirements engineering and design ergonomics. These works are mainly focused on the description of several categories of prioritization methods: nominal scales, ordinal scales and ratio scales [1]. Comparison of these methods deals with the evaluation on usability and effectiveness by designers (e.g., [2]) or on their costs-benefits (e.g., [3]); the time needed to prioritize or the user satisfaction are some others criteria (e.g., [1]). These works have in common to only address the requirements priorization by designers, not by users.

These studies neither aimed to analyze the consequences of the use of prioritization methods on design decisions, nor to analyze how the results of prioritization by future

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 1–15, 2013.

users could impact the design of the product or the software. Yet, it is now clear that user involvement in design is beneficial to provide a better balance between the designed artefact and the users' needs [4]. This is true in standard product design as much as in software design. This remains the same in a user-centered perspective, where the user is involved in the establishment of the needs and / or evaluation of the artefact (e.g., [5]), as in an approach called "participatory design" which considers the user as a co-designer involved in design decisions (e.g., [6]). This question underlines the issue of the involvement of several user profiles in the design process. While several studies have shown the interest to involve novice users and expert users in design (e.g., [7]), few empirical studies have sought to demonstrate the interest of involving end-users with different constraints, works and backgrounds, in the design process and even less in the requirements prioritization activity.

The main objective of this paper is to describe an empirical study on the involvement of different user profiles to the requirements prioritization. For that, we will firstly examine the real contribution of different profiles of end-users of a virtual reality (VR) software to the prioritization of functionalities. Because users can have different priorities according to their job and background, three profiles of users have been studied comparatively (stylists, engineers and marketers) relatively to the use of a nominal method of prioritization.

In the next section, we present a further review of the literature on requirements prioritization by different stakeholders, included end-users of a system, in the literatures of requirements engineering and ergonomics. Then we describe the methodology of our empirical study, and we expose our results which characterize the activity of prioritization of three users' profiles of a VR system and its consequences on the design choices.

2 Requirements Prioritization in Design by End-Users

2.1 The "Requirement Prioritization" Concept According to Stakeholders

Requirement prioritization consists in assigning different priorities to the requirements in order to obtain a relative order between them [8] and finally to determine which requirements should be implemented first [9]. In that context, the requirements prioritization is a design activity [10]. This prioritization activity takes place in the selection stage which precedes the technical realization [11]. From this viewpoint, requirements prioritization is a crucial step during the software process.

In requirements engineering, the selection is made by the requirement engineer or the project manager or even the developer himself, based on four recommended criteria:

- the technical feasibility of each alternative [11];
- the degree of uncertainty and risk associated with each alternative [12];
- the evaluation of the costs and benefits of each alternative [13];
- the degree of convergence of different stakeholders in a design project for each alternative [11].

In this literature, the end user does not participate directly in the selection of alternatives, contrary to others disciplines where prioritization can be performed by

end-users. In ergonomics, the selection of alternatives is done – at least partly – after the production phase of requirements and candidate specifications. Selected alternatives are used to produce a model which could evolve, that is, not directly the final software as in requirement engineering. In ergonomics as in requirement engineering, the stages of selection of the alternatives are prerequisites for achieving technical realization [14].

Both in the case of an iterative user-centred design [5] than in the case of participatory design [4] results obtained from the evaluation of the model are elements that will help to (re-)orient the design choices for a prototype. Indeed, users are most concerned to provide designers the necessary criteria to justify the new design choices in terms of destination of the artefact, but also in terms of profits and benefits for them. Moreover, the involvement of users is higher in a context of participatory design in which users, as co-designers, have the opportunity to make design decisions as well as designers. This suggests that decisions will take into account, at least in part, benefits for users, and not only designers' constraints as in requirement engineering. Loup-Escande demonstrated that the backers' requirements and users' needs, moderated by the designers' constraints, should be the specifications for the system design [15].

2.2 Prioritization Methods and Tools

There are many approaches, tools and methods recommended or used for prioritizing such as nominal scales, ordinal scales and ratio scales [2] which allow people to assign qualitative or quantitative values to requirements.

The 'Attributed Goal-Oriented Requirements Analysis' method (AGORA) [16] is an example of prioritization method, based on a graph used to decompose misunderstood goals until their understanding by each project' participants. In nominal scale methods, requirements are assigned to different priority groups. An example is the MoScoW method, which consists in grouping all requirements into four priority groups, that are requirements that the project (must / should / could / won't) have. All requirements listed in a category are of equal priority, which does not allow a finer prioritization. Ordinal scales methods produce an ordered list of requirements. For example, the simple ranking where the most important requirement is ranked 'one' and the least important is ranked 'n'. Simple sorting algorithms of sorting are also well suited to the requirements prioritization: for example, the "bubble sort" algorithm permits, by comparing the relative importance of requirements in pairs, to obtain a list of ordered requirements [17]. Another known method called Analytic Hierarchy Process asks users to compare all pairs of requirements [18]. Ratio scale methods provide relative difference between requirements (e.g. the hundred dollars method ask users to allocate a sum of money at each requirement). In addition to an ordered list of requirements, this method also helps to know the relative importance of each requirement in relation to other ones.

Users of these methods are generally specialists. Also, methods such as the 'bubble sort' remain difficult to master by people from the general public [1]. Different studies compared these methods (e.g., [1]): the metric used is the performance of the user (e.g., the number of decisions to make, time spent for the prioritization) or his satisfaction. Nevertheless, these studies are focused on the designers and elude the users of software which will contain implemented requirements.

2.3 Empirical Studies Focused on Prioritization Methods by End-Users

Prioritization and selection of functionalities can in some cases be made in two stages. This approach is sometimes used in ergonomics and adopted in some design projects to develop innovative software. Requirements are firstly identified by the engineers with technical or cost constraints. Then, a panel of people representing the users is asked to prioritize the requirements according to their own criteria. This preliminary selection of the functionalities by the designers, prior to their prioritization by users, can result in the removal of functionalities with a potential high added value for users, because perceived as complex or costly for designers.

In other projects, requirements prioritization and selection can be performed by a group composed of designers and users. The integration of users in the selection of design choices helps to develop software in which functionalities and properties of the artefact will bring a real added value to users. We present two studies that include end users in requirements prioritization [19, 20]. These two studies did not explicitly identify the role of end users in the prioritization task: they were included in the group as well as designers and backers. But priority requirements (i.e. judged important) are different, according to the role or the status of the project stakeholders [8].

In the first study [20], participants (two users, three therapists, two technicians and a stylist) had to prioritize functionalities and select relevant functionalities to develop technical assistance for person with motor disability. They prioritized functional specifications by level of importance and of flexibility. This prioritization aimed at selecting the "relevant" features by performing a retrospective subjective assessment of them.

The second study aimed to clarify the problems and the needs of different stakeholders affected by repetitive strain injury in office activities [19]. To do this, focus groups involving ten participants were organized, composed of various profiles (e.g., users and managers of private/public companies, ergonomics experts …). The facilitator asked participants to list problems and needs related to the theme of 'repetitive strain injury and physical organization of the post', and other topics that they considered important to broach. Then, each focus group had to review the main requirements and prioritize them. Thus, participants were able to define by themselves the most important elements in terms of requirements, which avoided a personal interpretation by the facilitator of the previous discussions.

In both studies described above, the objective was not to analyze the activity of requirements prioritization by users specifically, but rather realized by a design group including users. The question of the impact of prioritization on design decisions is also not addressed. The limits of these previous empirical studies constitute the interest and motivation of the study described in this paper.

3 Methodology

To examine the requirements prioritization by different profiles of end-users and to evaluate the implications for the design, we analyzed the lists of functionalities prioritized on the basis of a questionnaire and the functionalities evoked spontaneously by users, in the context of a real software design project.

3.1 Context: The Project 3D Child

This exploratory study was conducted on an industrial project named '3D Child'. This project, led by a group of companies specialized in accessories for children, was divided into two parts. This paper covers the second sub-project in which we were involved. It was entitled '3D environments and places': the aim was to design a virtual reality tool for three SMEs (A, B and C) to help them to evaluate their future products and to reduce time and cost of designing industrial products especially in the preliminary phases of design (i.e. prototypes). The company A, with three sites, is a furniture manufacturer with 1,050 employees. The company B, with a presence in fifteen countries, designs baby products and employs 4,700 employees. The company C is a cabinet maker, specializing in toys and employing nine cabinetmakers. This sub-project resulted in the development of a software dedicated to the presentation of interactive 3D scenes (children bedroom and car) composed of future products in which human characters - modelled in 3D could move (see Figure 1). This virtual reality tool is named 'Appli-Viz'3D' and is a software made for decision support in industrial design.

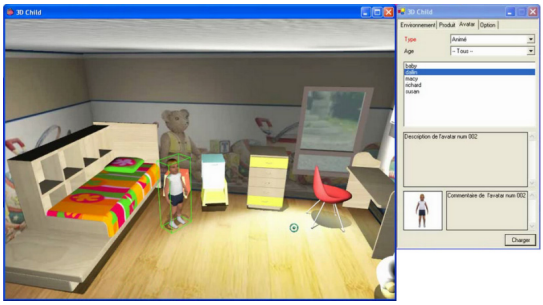


Fig. 1. Avatar in the " bedroom" 3D scene (from Appli-viz'3D)

3.2 Participants

This study involved twenty participants, who are end-users of Appli-Viz'3D: eight engineers (three from company A and five form company B), eight stylists (four from company A, three from company B and one from company C) and four marketers (two from company A and two from company B). Most users of Appli-Viz'3D were also designers in their firms (engineers and stylists). Therefore, they were able to know or understand the constraints and technical potentialities of the software, compared with users who are not designers. This has certainly facilitated the elicitation of needs (Reich et al., 1996). Participants were 41.3 years old on average (S.D. = 7.8; min = 28; max = 60) and 20.1 years of work experience (S.D. = 8.6; min = 5; max = 37).

3.3 Data Collection: Questionnaire and Nominal Scale Method

The material used to collect data regarding the process of the functionalities selection and prioritization was a questionnaire. The questionnaire contained a list of functionalities of Appli-Viz'3D that users had to prioritize (Table 1 – white part).

These functions were taken up from a previous demonstrator named 'Virtual Kid' - an online store that helped to launch the project '3D Child'. This questionnaire was elaborated by non-engineer designers, because we thought a priori that the designer-engineer would restrict at once the range of possibilities.

Participants had to prioritize these functionalities using marks from one to five (one = very important, two = important, three = moderately important, four = unimportant, five = useless). We chose the nominal method of prioritization for its ease of use that required no learning from users. We also asked participants to add their proposals of new features for the future tool.

The filling of the questionnaire was made following a presentation of the 3D Child project and a film showing the Virtual Kid application and the presentation of its functionalities. Virtual Kid movie should allow users to visualize the early functionalities on the list given in the questionnaire. In each company, we grouped the participants in the same room during the filing of the questionnaire.

3.4 Collected Data

Collected data are prioritized lists of functionalities, and eventually spontaneously evoked ones. These last functionalities can be prioritized or not. In the example shown in Table 1, spontaneously evoked functionalities (in gray) were not prioritized.

Table 1. List of functionalities completed by a stylist (*priority levels* in *green*, *additional functionalities* in *grey*)

1	Positioning 3D avatars with ergonomic postures
2	Moving in the scene
3	Changing the arrangement of the objects
2	Seeing the scene with the perspective of a child (small size)
2	Defining the dimensions of the environment
5	Changing the color of the objects
5	Changing the texture of the objects
3	Changing the scenery
2	Seeing avatars moving in the scene
3	Changing the color of the walls
	Allowing the avatar to interact with objects (climbing up a ladder, opening a drawer)
	Integrating ambient object directly linked to the function of the project

3.5 Analysis Method

To answer to our problematic, we analyze our data regarding:

- the relations between priority levels and functionalities;
- the relations between priority levels and proposals of additional functionalities;
- the characterization of the additional functionalities;
- the consideration of prioritization results by designers.

Statistical Analysis of the Priority Levels According to Functionalities. The quantitative analysis is based on numbers counting. We counted the occurrences of the priority level given by participants for each functionality (e.g. 2 = important, see section 3.3.). We added these numbers for each of the three users' profiles, then for all profiles taken into account (engineer, stylist and marketer). The resulting data tables are contingency tables are crossing the variable 'functionality' with the variable 'priority level' for each profile. We defined the overall strength of the link between two variables (e.g., priority level) by calculating the Cramer's V2. The link is considered strong for V2 between 0.16 and 1.0, weak for V2 lower than 0.04 and intermediate between the two. Then we characterized the local strength of the link between two modalities of these two variables (e.g., very important …) by computing relative deviations (RD) between modalities. There is attraction (i.e., similarities between variables) when the RD is positive and repulsion (i.e., disparities between variables) when it is negative. The attraction is said to be remarkable for a RD greater than 0.25. For full theoretical demonstration, see [21]. RD are often used in exploratory studies since they allow a measurement of local associations within a set of data (e.g., [22]).

Relations between Priority Levels and the Presence versus the Absence of Additional Functionalities. To analyze the relation between the priority levels associated with early functionalities and the proposal or the absence of additional functionalities, we identified the most frequently used priority level in each list (that is, the frequency of occurrence of this level was one point higher than the frequencies of other levels). When no level was distinguishable in terms of frequency, we removed the lists in question (three lists have been removed). A total of seventeen lists were analyzed. Then we counted the lists based on most frequently assigned levels and the presence vs. the absence of additional functionalities.

Analysis of the Proposals of Additional Functionalities. We studied the content of spontaneously evoked functionalities to determine if they were really new or only precisions of the functionalities already listed. We define new functionalities as they don't are a part of a functionality previously anticipated. A precision corresponds to a part or a detail of functionality already proposed in the list (e.g., "*integrating avatar of children and adults*" is a precision of "*positioning 3D avatars with ergonomic postures*" because it indicates that among the 3D models, adults and children should be integrated).

Then we counted the number of new functionalities and precisions to analyze the distribution in features spontaneously evoked. To avoid counting twice a same functionality, we analyzed qualitatively the functionalities added by users to identify those that were the same among all the lists. To do this, we established equivalences between expressions of functionalities that contained the same terms or synonyms (e.g., "*being able to manipulate and the functions of the products*" (evoked by an engineer) and "*permitting to use and to grasp objects*" (expressed by a marketer)).

Consideration of the Prioritizations by Designers. To analyze the consideration of the users' prioritization by designers, we tried to identify, among the prioritized functionalities, those which were validated and implemented by the designers. For this, we mapped the priority levels associated with early functionalities and requirements that have actually been really implemented in the project.

4 Results

All the results of our study, corresponding to the analysis methods presented previously, are synthesized in the following part then discussed in the fifth section.

4.1 Priority Functionalities for Users Not Systematically Implemented by Designers

The analysis of the relations between the two variables 'functionality' and 'priority level' shows an overall intermediate link (V2 = 0.07).

Table 2. Remarkable attractions between *functionalities* and *priority levels* (*X*), and state of the functionality at the end of the project (*grey* = not implemented)

Functionalities	**Useless**	**Unimportant**	**Md. important**	**Important**	**Very important**
Positioning 3D avatars with ergonomic postures					X
Moving the scene					X
Changing the arrangement of the objects				X	
Changing the scenery			X	X	
Defining the dimension of the environment	X				
Changing the color of the walls	X	X	X		
Seeing the scene with the perspective of a child (small size)					
Changing the color of the objects	X		X		
Changing the texture of the objects	X		X		
Seeing avatars moving in the scene		X			

Table 2 summarizes the functionalities (left column), remarkable attractions (based on relative deviations) between priority levels and functionalities, and the state of the functionality at the end of the project, implemented or not implemented (grey). For one functionality (i.e., "*seeing the scene with the perspective of child (small size)*"), no remarkable attraction was observed (blank line in the table 2).

This table underlines the absence of direct link between priorities associated by users and the state of the functionalities at the end of the project (implemented or not). For instance, the functionality "*positioning 3D avatars with ergonomic postures*", frequently associated with the 'very important' priority level (RD = 1.71) was not fully implemented. The reason given by the designer-engineer was related to technical constraints, particularly in terms of collisions between the avatar and the 3D model of the product. On the user side, some frustration has been felt to the delivery of the artefact. Conversely, the functionality "*changing the color of the walls*", which is characterized by strong attractions with priority levels 'moderately important' (RD = 0.84), 'unimportant' (RD = 0.50) and 'useless' (RD = 0.82) has been implemented. Finally, some features judged essentially 'moderately important' and 'useless' have not been implemented (e.g., "*changing the color of the objects*").

4.2 Different Priority Levels According to Users' Profiles

The analysis of the relations between the two variables 'functionality' and 'priority level' shows a strong overall association (V2 = 0.20) for the engineer profile, an intermediate link (V2 = 0.11) for the stylist profile and a strong overall link (V2 = 0.24) for the marketer profile.

We detail the priority levels associated with functionalities for each user profile in Table 3, based on analysis of relative deviations (i.e., remarkable attractions are represented by colors).

A first observation is that several priority levels may be associated with functionality within a same profile, as shown in Table 3: for example, the functionality "moving *the scene*" was associated with 'unimportant' and 'very important' levels by stylists.

A second observation is that the results suggest that the priority levels associated with a same feature may be common or specific depending on the profile of users (see Table 3). For example, priorities given for the functionality "*changing the texture of the objects*" are quite similar among profiles of users, contrary to priorities assigned to the functionality "*changing the color of the objects*".

Putting in perspective priority levels mostly assigned by each profile and functionalities really implemented suggests that there is no qualitative relation between the implementation of a functionality and the profile that judges it very important or important.

Table 3. Remarkable attractions between priority levels associated to each functionality, according to the profile (*E* = Engineers, *S* = Stylists, *M* = Marketers)

Functionalities	Profile	Useless	Unimportant	Mod. important	Important	Very important
Positioning 3D avatars with ergonomic postures	E					
	S					
	M					
Moving the scene	E					
	S					
	M					
Changing the arrangement of the objects	E					
	S					
	M					
Changing the scenery	E					
	S					
	M					
Defining the dimension of the environment	E					
	S					
	M					
Changing the color of the walls	E					
	S					
	M					
Seeing the scene with the perspective of a child (small size)	E					
	S					
	M					
Changing the color of the objects	E					
	S					
	M					
Changing the texture of the objects	E					
	S					
	M					
Seeing avatars moving in the scene	E					
	S					
	M					

4.3 The Users Who Assigned 'Important' and 'Unimportant' Priority Levels Evoked Additional Functionalities

The majority of the lists contain proposals of additional functionalities (12/17, 71% - 3 lists have been removed, see part 3.5.2).

The analysis of the relations between the two variables 'presence or absence of proposal of additional functionalities' and 'priority level' shows an overall intermediate link (V2 = 0.13). The analysis of the relative deviations reveals that the lists of functionalities with the most of 'very important' priority levels don't contain any proposal for additional functionalities (RD = 0.36). For example, a stylist who assigned seven times the level 'very important' did not add any extra functionality. Similarly, the lists in which the priority level the most frequently assigned is 'moderately important' does not contain any functionality additions (RD = 0.36).

However, when the level 'important' is the most used in a list, this one also contains proposals for additional functionalities (RD = 0.42). Similarly, the level 'moderately important' is characterized by a strong attraction with the 'proposal of additional functionalities' (RD = 0.42).

When the level 'useless' is the most used in the lists, they may as well include proposals for additional functionalities, such as no proposals. For example, a stylist who assigned five times the mark 'five' (useless) did not propose additional functionalities. Conversely, an engineer proposed three additional functionalities whereas he mainly attributed the mark 'five' (i.e., useless) to the early functionalities.

4.4 The Spontaneously Evoked Functionalities are Mainly Precisions of Anticipated Functionalities

Our data show that thirteen out of twenty participants evoked additional functionalities: seven are engineers, three are stylists and three are marketers.

The thirty-four additional functionalities evoked by users are distributed as follows: thirteen new functionalities (62%) and twenty-one precisions (38%). These thirty-four functionalities are the result of the sum of functionalities added in each list.

After grouping similar functionalities between two or more lists, we obtained finally fifteen additional functionalities: six new ones and nine precisions. The distribution of the six new functionalities and nine precisions in terms of common vs. specific to different user profiles is presented on the Figure 2.

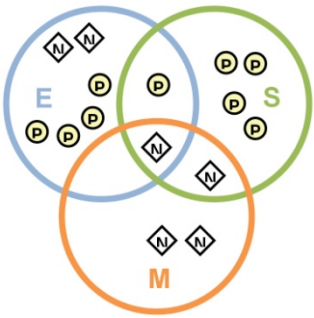


Fig. 2. Distribution of new functionalities (*N*) and precisions (*P*) common or specific to each profile (*E* = Engineer, *S* = Stylist, *M* = Marketer)

As shown in Figure 2 concerning the six new functionalities (N), we observe that:

- One functionality is common to engineers, stylists and marketers (e.g. '*allowing the avatar to interact with objects (climbing up a ladder, opening a drawer)*', expressed by a designer);
- One functionality is common to stylists and marketers (e.g. '*making a short movie […]*', evoked by a designer);
- Two functionalities are specific to marketers (e.g. '*zooming on specific functions of the furniture during their use*');
- Two functionalities are specific to engineers (e.g. '*measuring spaces on the product*').

Concerning the nine precisions (P), we observe that:

- One functionality is common to stylists and engineers (e.g. '*positioning avatars of children and adults in a same scene […]*', evoked by a stylist; *'positioning the child/parent pair*', expressed by an engineer);
- Four functionalities are specific to engineers (e.g. '*defining standard environments: car trunk (Mini format), train door, bus, sidewalk, supermarket checkout*');
- Four functionalities are specific to stylists (e.g. '*integrating typical objects: baby's bottles, changing tables … in the scene to assess the function of the furniture*').

5 Discussion

The results show that the priority levels associated with the same functionalities can be common to several user profiles or different according to user profiles. We note that several priority levels can be assigned to one functionality by a same user profile. This confirms the need to develop consultations between several profiles of end users, followed by consultations between designers and end-users [23].

We observe that no marketer evaluated a functionality as useless, contrary to engineers and stylists who attributed the level 'useless' to some functionalities. One possible explanation is that engineers and stylists, who are more familiar with making design choices, are more at ease to say that a feature is useless for a given artefact. Marketers seem to prefer to assume that all the functionalities anticipated by the designers are useful at first sight, because none of them assigned the priority level 'useless' to the functionalities.

Our data highlight that a functionality considered as 'very important' for users might not be implemented as is. However, knowing that this functionality was 'very important' for users has driven designers to find a compromise leading to the implementation of a part of the functionality. Conversely, functionalities generally considered as 'unimportant' or 'useless', but considered as 'very important' by a profile of users in particular, may have been implemented. These findings suggest that the functionalities prioritized by the users are a source of information for designers. Theses ones try to take them into consideration: they care about the usefulness of the software. But user priorities are a set of information among others. That leads designers to implement functionalities they consider less costly (in financial and temporal

terms), without leaving aside the prioritizations of the users which represent more a base of exchange than a list of functionalities to implement as they are. This confirms the interest to compose multidisciplinary team, including several profiles of end-users, in the early stages of the design process [24] and not only in the evaluation phases.

A last result of our study is that the majority of users evoked additional functionalities following the prioritization task, especially when users have attributed most frequently the levels 'important' or 'unimportant'. This evocation of post-prioritization requirements confirms what was supposed by [19] concerning the interest of involving users in the prioritization to clarify or propose new features, beyond their priorities. We suppose that a possible explanation is that providing users with some examples of functionalities enable users to have an initial understanding of the technological potential of the artefact. Prioritizing these examples allows them to imagine what could be the future artefact, and to evoke additional functionalities. These additional functionalities were mostly precisions of functionalities already anticipated and, to a lesser extent, entirely new functionalities. These last ones were common to several user profiles (i.e. either common to engineers, stylists and marketers or to stylists and marketers) or specific to a profile (in this case, marketers and engineers). The precisions were essentially specific to engineers and stylists.

6 Conclusions and Perspectives

The phase of requirements prioritization step is a key element of the software process as it will lead to the selection of functionalities to implement. The results presented above, and the ensuing discussion, allow us to make several recommendations to promote the design of artifacts with a real added value to the user. A first prerequisite is the integration of end users in the prioritization phase. In the field of collaborative engineering, previous studies have demonstrated that multidisciplinary design teams are beneficial to the design of products. The results of the study related in this paper allow us to go further by claiming that multidisciplinary teams of users are beneficial to the design of products which are in fact useful for them. Indeed, users have additional needs related on their job profile, which results in different priorities. This is particularly interesting in a context of "participatory" design, developed in Living Labs. Participatory design is based on a strong involvement of users in the expression of needs or the imagination of solutions, and on the fact that users must make decisions as well as designers. The prioritization phase is essential, because it allows future users to imagine new functionalities that were not proposed by the designers. However, giving users a first list of functionalities is crucial for them to imagine the future artifact to have food for thought. These prioritized lists are necessary for to allow designers to consider both their own constraints and user needs. This leads them to make compromises that benefit the real utility of the artifact to be designed.

A limitation of our study concerns the small project and the small sample size. That justifies the exploratory status of this study which allow us to obtain trends and not general conclusions about the differences between stakeholders. A second limit concerns the absence, in our data collection protocol, of elicitation interviews

following the filling of the questionnaire. The realization of such interviews has not been possible because of constraints concerning the availability of the participants.

From this limit, a research perspective is to analyze the requirements prioritization by users adding to our original protocol elicitation interviews to know the reasons for assigning a priority level and how users perform this task. This would allow to identify finely subjective criteria justifying the levels assigned to each functionality. For example, we would then be able to explain why a user who gave the level 'unimportant' to functionality: is it because he imagined a very infrequent use or because he guessed he wouldn't need this functionality (but he did not dare to give the level 'useless')? We would also understand how each assignment was performed. Thus, we could know if people gave 'very important' level first or if they began with 'useless' functionalities. This would show that users know immediately what would useless or instead that users a priori know what they would need.

References

1. Ma, Q.: The Effectiveness of Requirements Prioritization Techniques for a Medium to Large Number of Requirements: A Systematic Literature Review. In: Auckland University of Technology as a Part of the Requirements for the Degree of Master of Computer and Information Sciences. School of Computing and Mathematical Sciences (2009)
2. Karlsson, J., Wohlin, C., Regnell, B.: An evaluation of methods for prioritizing software requirements. Inform. Software Tech. 39, 939–947 (1998)
3. Christian, T., Mead, N.R.: An Evaluation of Cost-Benefit Using Security Requirements Prioritization Methods. U.S. Department of Homeland Security (2010)
4. Caelen, J.: Conception participative par « moments »: une gestion collaborative. Le Travail Humain 72, 79–103 (2009)
5. Bastien, J.M.C., Scapin, D.: La conception de logiciels interactifs centrée sur l'utilisateur: étapes et méthodes. In: Falzon, P. (ed.) Ergonomie. PUF, Paris (2004)
6. Von Hippel, E.: Democratizing Innovation. MIT Press (2005)
7. Popovic, V.: Expert and Novice Users Model and their Application to the Design Process. In: Journal of the Asian Design International Conference (2003)
8. Berander, P., Andrews, A.: Requirements Prioritization. In: Aurum, A.W.C. (eds.) Engineering and Managing Software Requirements. Springer (2005)
9. Iqbal, A., Kahn, F.M., Khan, S.A.: A critical analysis of techniques for requirement prioritization and open research issues. International Journal of Reviews in Computing 1, 8–18 (2009)
10. Darses, F., Falzon, P., Béguin, P.: Collective design processes. In: International Conference on the Design of Cooperative Systems, pp. 141–149. INRIA (1996)
11. Alenljung, B., Persson, A.: Portraying the practice of decision-making in requirements Engineering - A case of large scale bespoke development. Requir. Eng. (2008)
12. Aurum, A., Wohlin, C.: The fundamental nature of requirements engineering activities as a decision-making process. Inform. Software Tech. 45, 945–954 (2003)
13. Macaulay, L., Fowler, C., Kirby, M., Hutt, A.: USTM: a new approach to requirements specification. Interacting with Computers 2, 92–118 (1990)
14. Maguire, M., Bevan, N.: User Requirements Analysis: A Review of Supporting Methods. In: Proceedings of the IFIP 17th World Computer Congress - TC13 Stream on Usability: Gaining a Competitive Edge. Kluwer, B.V. (2002)

15. Loup-Escande, E.: Vers une conception centrée sur l'utilité: une analyse de la co-construction participative et continue des besoins dans le contexte des technologies émergentes. Thèse de Doctorat, Université d'Angers (2010)
16. Kaiya, H., Horai, H., Saeki, M.: AGORA: attributed goal-oriented requirements analysis. method. In: IEEE Joint International Conference on Requirements Engineering, pp. 13–22. IEEE Press (2002)
17. Aho, A.V., Ullman, J.D., Hopcroft, J.E.: Data Structures and Algorithms. Addison Wesley (1983)
18. Saaty, T.L.: Analytic Hierarchy Process. Encyclopedia of Biostatistics. John Wiley & Sons, Ltd. (2005)
19. Collinge, C., Landry, R.: Prévention des troubles musculosquelettiques associés à la bureautique: analyse des besoins et portrait de la formation et de l'information (1997)
20. Plos, O., Buisine, S., Aoussat, A., Dumas, C.: Analysis and translation of user needs for assistive technology design. In: International Conference on Engineering Design, ICED 2007, p. 12 (2007)
21. Bernard, J.-M.: Analysis of local or asymmetric dependencies in contingency tables using the imprecise Dirichlet model, Lugano, Switzerland. Paper presented at the 3d International symposium on imprecise probabilities and their applications, ISIPTA 2003 (2003)
22. Anastassova, M., Burkhardt, J.-M., Mégard, C., Ehanno, P.: Results from a user-centred critical incidents study for guiding future implementation of augmented reality in automotive maintenance. Int. J. Ind. Ergonom. 35, 67–77 (2005)
23. Reich, Y., Konda, S.L., Monarch, I.A., Levy, S.N., Subrahmanian, E.: Varieties and issues of participation and design. Des. Stud. 17, 165–180 (1996)
24. Tichkiewitch, S., Tiger, H., Jeantet, A.: Ingénierie simultanée dans la conception de produits. In: Universités d'été du pôle productique Rhône-Alpes (1993)

ACME+: An ADL for Quantitative Analysis of Quality Attributes

Imen Derbel[1], Lamia Labed Jilani[1], and Ali Mili[2]

[1] Institut Superieur de Gestion, Bardo, Tunisia
[2] New Jersey Institute of Technology, Newark, NJ, 07102-1982, U.S.A.

Abstract. One of the main issue of software systems engineering is determining the overall system quality attributes at an early stage. This has several advantages such as early detection of problems, cost benefits and assuring that the chosen architecture will meet both functional and non-functional quality attributes. One emerging approach for dealing with such early analysis is to evaluate the system quality attributes at the architectural level. However, there is a notable lack of support for representing and reasoning about non functional attributes in existing Architectural Description Languages(ADLs). In this paper, we propose Acme+ as an extension of Acme ADL and discuss its abilities to represent and evaluate non functional attributes such as response time, throughput, failure probability, etc. We also describe a tool that reads a text file containing an architecture described in ACME+ and outputs the quality attributes of all the system and detects the component or the connector bottleneck.

Keywords: Software Architecture Analysis, Quality Attributes, Architecture Description Language, Performance, Reliability, Acme, Bottleneck.

1 Introduction

The concept of software architecture has emerged in the eighties as an abstraction of the design decisions that precede functional design, and pertain to such aspects as broad system structure, system topology in terms of components and connectors, coordination between system components, system deployment, and system operation [17,8]. This concept has gained further traction through the nineties and the first decade of the millennium, by virtue of its role in many modern software engineering paradigms, such as domain engineering, product line engineering. Whereas functional design and programming determine the functional attributes of a software product, the architecture of a software product determines its non-functional attributes, i.e. properties such as: response time, throughput, failure probability, etc; we refer to these as quality attributes of the software product. A number of architecture description languages (ADL's) have emerged in the past two decades, including ACME (CMU) [16], Wright (CMU)[3], Rapide (Stanford University) [25], PADL (Urbino) [1,2]. Even though many of these languages embody state of the art ideas about software architectures, and despite the importance of non functional attributes in the characterization of software architectures, to the best of our knowledge none of these ADL languages offers automated support for analyzing quality attributes of software architectures. In this paper we propose to fill

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 16–32, 2013.

this gap by proposing an ADL which is a modified version of ACME (we refer to this language as ACME+), and building a compiler for this language, with the following characteristics:

- ACME+ is based on ACME's architecture ontology, in that it represents architectures in terms of components, connectors, ports and roles.
- It uses ACME's `property` construct to represent the quality attributes of components and connectors; but while ACME considers the data entered under `property` as a mere comment, which it does not analyze, we give it a precise syntax and use it in our analysis.
- Whereas ACME lists the ports of a component and the roles of a connector, and does not specify any relation between the ports of a component or the roles of a connector, we introduce special purpose constructs that specify these relations, and use them in our analysis.

Among the questions that we envision to address/answer, we cite the following:

- Given a set of values for the quality attributes of components and connectors, what are the values of the quality attributes of the overall system?
- How do the system-wide attribute values depend on component-level and connector-level values and how sensitive are system-wide attribute values to variations in component-level and connector-level values?
- Which component-level or connector-level attribute values are causing a bottleneck in system wide attribute values?

In section 2, we briefly present and motivate the main syntactic features that we have added to ACME; in section 3, we discuss the semantics of these constructs, in terms of Mathematica equations that we associate to them; in section 4, we discuss bottleneck analysis; in section 5, we show how we generate the proposed compiler; in section 6, we discuss related work. The paper concludes in section 7 by a discussion of our prospects for future research.

2 ACME+: Syntax

In order to enable us to represent and reason about non functional properties of software architectures, we need an architectural description language that offers the following features:

1. Support the ability to represent components, connectors, ports and roles.
2. Support the ability to represent quantitative non functional attributes of components and connectors.
3. Provide constructs that enable us to represent operational information that impacts the non functional attributes. At a minimum, we must be able to identify, among ports of a component (and roles of a connector) which ports are used for input and which ports are used for output. Furthermore, if we have more than one input port or more than one output port, we need to represent the relation between the ports: are they mutually synchronous or asynchronous? Do they carry duplicate information? or disjoint/ complementary information? or overlapping information?

4. Provide means for a component (or a connector) to represent more than one relation from input ports (roles) to output ports (roles). The reason we need this provision is that often the same component (or connector) may be involved in more than one operation, where each operation involves a different configuration of ports (roles), and have different values for its non functional attributes.

Among all the architecture description languages that we have considered, we have found none that meets these four requirements. Most languages devote much attention to representing the topology of the system; some languages, such as Wright [5] and PADL [1,2] complement the topological information with operational information, but the latter is expressed in CSP [19] which is too detailed for our purposes, and at the same time fails to always provide the information we need. To cater to the four requirements we have presented above, we adopt ACME's basic syntax and ontology, and add to it the concept of `functional dependency`.

2.1 ACME+: ACME Extension with Functional Dependency

We adopt ACME's ontology of components, connectors, ports and roles, and its main approach for representing software architectures. This approach represents components by describing a number of their properties, including a list of their relevant ports; and it represents connectors by describing their properties, including a list of their relevant roles [18]. Furthermore, ACME enables the architect to build arbitrary topologies by means of attachment statements, which connect ports to role and roles to ports. The ACME code below shows a simple ACME description of a client-server architecture:

```
System simpleCS = {
Component client = {Port call_rpc; };
Component server = { Port rpc_request; };
Connector rpc = { Role client_side;
                  Role server_side; };
Attachments = {
client.call_rpc to rpc.client_side;
server.rpc_request  to rpc.server_side; }}
```

In order to enable us to represent and reason about non functional properties of software architectures, we enrich ACME ADL with a new construct which must support the following capabilities:

- Specify the operational information of a software architecture that impacts the non functional attributes.
- Identify the input ports and output ports of a component, as well as the origin roles and destination roles of a connector.
- Represent relations between ports of the same type (input, output) and between roles of the same type (origin, destination).
- Represent non functional properties of components and connectors from a predefined catalog, using predefined units of measurement (e.g. milliseconds for response time, transactions per second for throughput, probability for failure probability, hours for MTTF (Mean Time To Fail), percentage for availability).

The BNF syntax of the proposed construct is defined as follows:

```
FuncDependency ::= FunDep ":" "{" Rdeclaration
                   "}" ";" ;
Rdeclaration  ::= Rdecl Rdeclaration | ;
Rdecl         ::= Identifier "(" Inputs ";"
                  Outputs ";" Properties ")";
Inputs ::= Input "(" InputSpecification ")" | ;
Outputs ::= Output "(" OutputSpecification ")" | ;
InputSpecification ::= InSelection "("
       InSynchronisation "(" ListId ")" Spec ")"
       | InSynchronisation "(" Identifier ")" ;
InSelection    ::=  AnyOf |  AllOf |  MostOf ;
InSynchronisation ::= Synchronous|Asynchronous ;
OutSelection ::= Duplicate|Exclusive|Overlapping ;
OutSynchronisation ::= Simultaneous|Asavailable ;
ListId    ::= Identifier "," ListId | Identifier ;
Spec      ::= "," InputSpecification Spec | ;
OutputSpecification ::= OutSelection "("
                        OutSynchronisation
                       "(" ListId ")" ")" | ;
Properties ::=Properties "(" PropSpecification ")"
              | ;
PropSpecification ::= PropSpecification PropSpec| ;
PropSpec       ::=  procTime "=" PTvalue ";"
                    | thruPut "=" TPvalue ";"
                    | failProb"=" FPvalue ";"
PTvalue        ::= Literal sec | Literal msec;
TPvalue        ::= Literal trans/sec
                    | Literal trans/min;
FPvalue        ::= Literal  ;
```

In the construct `FuncDependency`, **FunDep** is a reserved word which serves as a header indicating that the following descriptions pertain to functional dependency relations of the component in question. This construct consists of one or more functional relations (`Rdeclaration`). Each relation (`Rdecl`) is identified by a Relation Name and corresponds to a possible role played by the component. It connects Input Ports (`Inputs`) and Output Ports (`Outputs`) and is characterized by non functional properties (`Properties`) such as processing time, throughput, and failure probability. The term **Input** is a reserved word which indicates the list of input ports and their operational information through `InSelection` and `InSynchronisation` constructs. The term `InSelection` indicates whether all of the input ports are needed (**AllOf**), or any one of them is sufficient (**AnyOf**), or most of them are needed (**MostOf**), as would be the case in a modular redundancy voting scheme for example. The term `InSynchronisation` indicates whether the ports have to make data available **Synchronously** or **Asynchronously**. Similarly the term **Output** is a reserved word which indicates the list of output ports and their operational information through `OutSelection` and `OutSynchronisation` constructs. The term `OutSelection` indicates whether the outputs posted on the different output ports are **duplicate**, **exclusive**

or **overlapping**. The term $OutSynchronisation$ indicates whether the data posted on output ports is posted simultaneously on all output ports (**Simultaneous**), or is posted as available (**Asavailable**). The term **Properties** is a reserved word which indicates the non functional properties of the component. So far, we have restricted the property names to **procTime**, **thruPut** and **failProb**. For each property, we specify its value respectively through the terms `PTvalue`, `TPvalue` and `FPvalue`. The above rules form the basis of the proposed extensions. The rules can, however, be extended to include multiple requirements and information where necessary. In order to illustrate the proposed extensions in practice, we present an example in the next section.

2.2 A Sample Example of an Architecture Description with ACME+

To illustrate how the proposed construct works, we consider the architecture of the Aegis Weapons System [5].

Figure 1 depicts the basic architecture of Aegis represented in ACME Studio. The system consists of seven components: Geo_Server, Doctrine_Reasoning, Doctrine_Authoring, Track_Server, Doctrine_Validation, Display_Server and Experiment_Control. To this configuration, we add, for the sake of illustration, two dummy components Sink and Source and their associated connectors. Using our proposed constructs of functional dependency, we give below examples of ACME+ description. For the sake of brevity, we content ourselves with giving ACME+ descriptions of only one component of Aegis system. The overall architecture description of the Aegis Weapon System in ACME+ is available online at:
http://web.njit.edu/ mili/AegisArch.txt.
The description relative to Display_Server component is defined by:

```
Component Display_Server {
Port inPort0; Port inPort1;
Port inPort2;Port inPort3;
FunDep = {R(
Input(AllOf(Synchronous(inPort0; inPort1;
                        inPort2; inPort3)));
Output(outPort);
Properties(procTime=1;thruPut=0.4;failProb=0.3)
)}};
```

Display_Server operates in only one task (`R`) that requires all of data provided by the input ports synchronously in order to display results on its output port. This task is characterized by quality attributes defined in terms of processing time, throughput and failure probability. To test the adequacy of this languages, we have used it to represent a number of sample architectures, including the Video Animation Repainting System [10],and the Rule Based System [17]. In all cases we find that the information required by the Acme+ description is readily available as part of the architectural description.

3 ACME+: Semantics

3.1 A Logical Framework

In order to use the information recorded in the proposed constructs for the purpose of analyzing software architectures, we take the following modeling decisions:

- Each port in a component is labeled for **inPort** or for **outPort**.
- Each role in a connector is labeled as a **fromRole** or a **toRole**.
- Each architecture has a single component without input port, called the $Source$, and a single component without output ports, called the $Sink$.

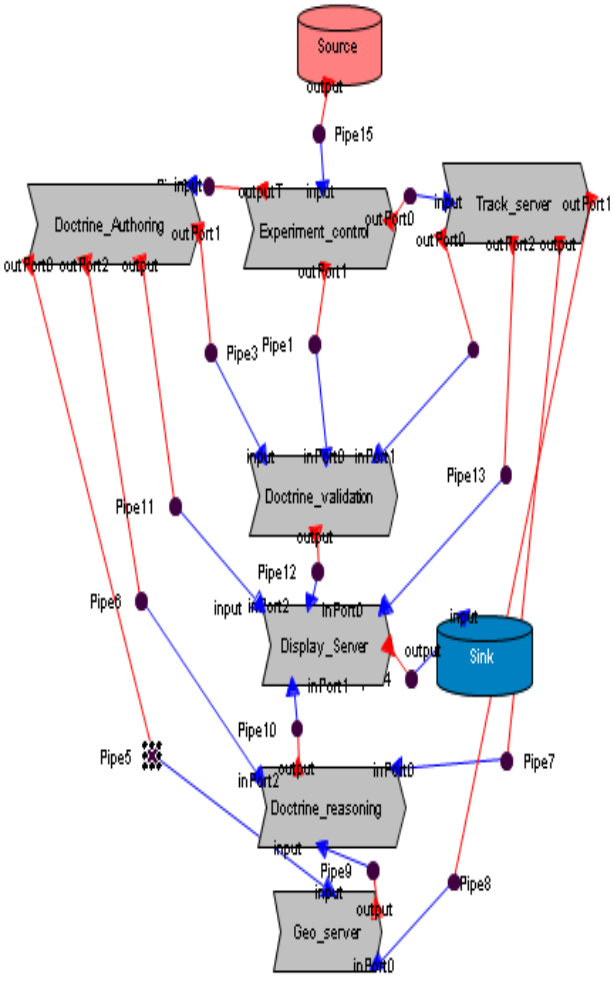


Fig. 1. Aegis system architecture represented in ACME Studio

In this discussion, we are interested in three sample non-functional attributes, namely: (1) Response time, measured in milliseconds. We assume that each component has a property of type real called **procTime** that represents the component's processing time and each connector has a property of type real called **transTime** that represents the connector's transmission time. (2) Throughput, measured in transactions per second. We assume that each component and each connector has a property of type integer called **thruPut**. (3) Failure probability, measured as a probability. We assume that each component and each connector has a property of type real called **failProb**. We define the system wide attributes as:

- For each port and each role, we assign a set of attributes that are related to the quality attributes we are interested in. Hence each port has a response time attribute called RT, a throughput attribute called TP, a failure probability attribute called FP (for failure probability). We distinguish between component and connector *properties*, which are specified in the ACME+ source code, and the (similar sounding but distinct) port and role *attributes*, which are assigned to ports and roles by our attribute grammar, and are computed by our compiler.
- For the output port of the source component, we assign trivial values for these attributes, such as zero for the response time, zero for failure probability, and infinity for throughput. We write:

$$Source.outPort.RT = 0. \tag{1}$$

$$Source.outPort.TP = \infty. \tag{2}$$

$$Source.outPort.FP = 0. \tag{3}$$

- For each functional dependency relation we associate an equation between the attributes of the ports and roles that are involved in the relation. The equation depends of course on the nature of the functional dependency; for example, if two ports are linked by an **AllOf** construct, the response time associated with the output ports of the relation is the maximum of the response times associated to the output ports to which we add the processing time of the components, and the throughput associated to the output ports is the minimum of the throughput associated with the input ports, and the throughput capacity of the component. This process is discussed in greater detail in the following section.
- The values of the non functional properties for the overall architecture are then the values of the relevant attributes for the input port of the sink component; hence the response time of the whole system architecture is $Sink.inPort.RT$; the throughput of the whole system architecture is $Sink.inPort.TP$; and the failure probability of the whole system architecture is $Sink.inPort.FP$. The values of these attributes are computed inductively from the properties attached to the components and connectors (**procTime**, **transTime**, **thruPut**, **failProb**). We write:

$$System.ResponseTime = Sink.inPort.RT. \tag{4}$$

$$System.Throughput = Sink.inPort.TP. \tag{5}$$

$$System.FailureProbability = Sink.inPort.FP. \tag{6}$$

3.2 Inductive Rules

Rules between Components and Connectors. Whenever a port of a component is attached to the role of a connector, their attributes are equated. For example, if the output port of component C is attached to the origin role of connector N, we write:

$$C.outPort.RT = N.fromRole.RT. \tag{7}$$

$$C.outPort.TP = N.fromRole.TP. \tag{8}$$

$$C.outPort.FP = N.fromRole.FP. \tag{9}$$

Single Input/ Single Output. The inductive rules are straightforward for components that have a single input port and a single output port, and for connectors that have a single origin role and a single destination role; we illustrate these rules on a connector. Given a connector N, we write an equation that links the attributes of the origin role (fromRole), the attributes of the destination role (toRole), and the properties of the connector. We write:

$$N.toRole.RT = N.fromRole.RT + N.transTime. \tag{10}$$

$$N.toRole.TP = Min(N.fromRole.TP; N.thruPut). \tag{11}$$

$$N.toRole.FP = 1 - (1 - N.fromRole.FP)\,(1 - N.failProb). \tag{12}$$

Multiple Inputs and Outputs. When a component has more than one input port or more than one output port, then the inductive rules within the component depend on the exact relation between the multiple ports of the same type (input, output). We review the main configurations for a component, and argue that similar rules apply for connectors. For each component, these equations link the values of the attributes at the input ports and output ports with the values of internal properties (**procTime**, **thruPut** and **failProb**). These equations depend on the nature of the functional dependency relations. We let C designate a component, whose input ports are called $inPort_1; ...; inPort_n$ and output ports are called $outPort_1; ...; outPort_k$. We suppose that these input and output ports are related with a functional dependency relation R expressed as follows:

```
R(
Input(InSelection(InSynchronisation
              (inPort1; ..; inPortn)));
Output(OutSelection(OutSynchronisation
            (outPort1; ..; outPortk)));
Properties(procTime=0.7;thruPut=0.2;failProb=0.2)
 )
```

We review in turn the three attributes of interest.

Response Time. For each output port $outputP_i$ expressed in the relation R, we write:

$$C.outPort_i.RT = function(C.inPort1.RT; \\ ...; C.inPort_n.RT) + C.R.procTime. \tag{13}$$

where *function* depends on the construct `InSelection`, expressing the nature of the relation between input ports. If `InSelection` is **AllOf**, then *function* is the maximum, we write:

$$C.outPort_i.RT = Max(C.inPort_1.RT; ...; C.inPort_n.RT) \\ +C.R.procTime. \tag{14}$$

If **InSelection** is **AnyOf**, then *function* is the minimum, we write:

$$C.outPort_i.RT = Min(C.inPort_1.RT; ...; C.inPort_n.RT) \\ +C.R.procTime. \tag{15}$$

If **InSelection** is **MostOf**, then *function* is the median, we write:

$$C.outPort_i.RT = Med(C.inPort_1.RT; ...; C.inPort_n.RT) \\ +C.R.procTime. \tag{16}$$

Throughput. For each output port $outPort_i$ of the component C expressed in the relation R, we write an equation relating the component's throughput and $inPort_i.TP$. This rule depends on whether all of inputs are needed, or any one of them. Consequently if `InSelection` is **AllOf**, and since the slowest channel will impose its throughput, keeping all others waiting, we write:

$$C.outPort_i.TP = Min(C.R.thruPut; \\ (C.inPort_1.TP + ... + C.inPort_n.TP)). \tag{17}$$

Alternatively, if `InSelection` is **AnyOf**, since the fastest channel will impose its throughput, we write:

$$C.outPort_i.TP = Max(Min[C.R.thruPut; C.inPort_1.TP]; \\ ...; Min[C.R.thruPut; C.inPort_n.TP]). \tag{18}$$

Failure Probability. For each output port $outPort_i$ of the component C expressed in the relation R, we write an equation relating component's failure probability and input ports failure probability. This rule depends on whether all of inputs are needed, or any one of them. We first consider that $inPort_i$ provide complementary information (`InSelection` is **AllOf**). A computation initiated at $C.outPort_i$ will succeed if the component C succeeds, and all the computations initiated at the input ports of C succeed. Assuming statistical independence, the probability of these simultaneous events is the product of probabilities. Whence we write:

$$\begin{aligned} C.outPort_i.FP = 1- \\ (1 - C.inPort_1.FP \times ... \times C.inPort_n.FP) \\ (1 - C.R.FailProb). \end{aligned} \tag{19}$$

Second we consider that $inPort_i$ provide interchangeable information (`InSelection` is **AnyOf**). A computation initiated at $C.outputP_i$ will succeed if component C succeeds, and one of the computations initiated at input ports $C.inPort_i$ succeeds. Whence we write:

$$\begin{aligned} C.outPort_i.FP = 1 - (1 - C.inPort_1.FP) \times ... \\ \times (1 - C.inPort_n.FP)(1 - C.R.FailProb). \end{aligned} \tag{20}$$

3.3 Illustration with an Example

We show below equations we write for Display_Server component of AEGIS architecture, along with the Mathematica equations that our compiler generates from the code proposed earlier. For the sake of brevity, we present equations written for only one component, and leave it to the reader to see how the rules for other components can be derived by analogy.

Within Component Display_Server. The compiler generates the following Mathematica equations for Display_Server:

$$\begin{aligned} DisplayServer.outPort.RT = Max(\\ DisplayServer.inPort0.RT; \\ DisplayServer.inPort1.RT; \\ DisplayServer.inPort2.RT; \\ DisplayServer.inPort.RT)+ \\ DisplayServer.R1.procTime \end{aligned} \tag{21}$$

$$\begin{aligned} DisplayServer.outPort.TP = Min(\\ DisplayServer.R.thruPut; \\ \sum_{i=0}^{3}(C.inPort_i.TP)) \end{aligned} \tag{22}$$

$$\begin{aligned} DisplayServer.outPort.FP = 1- \\ (1 - DisplayServer.R.failProb)\times \\ (1 - \prod_{i=0}^{3} DisplayServer.inPort_i.FP) \end{aligned} \tag{23}$$

Between Display_Server and Connectors. The compiler generates the following Mathematica equations between Display_Server input ports and connectors roles:

$$DisplayServer.inPort0.RT = Pipe13.toRole.RT \tag{24}$$

$$DisplayServer.inPort1.RT = Pipe10.toRole.RT \tag{25}$$

$$DisplayServer.inPort2.RT = Pipe12.toRole.RT \tag{26}$$

$$DisplayServer.inPort3.RT = Pipe11.toRole.RT \tag{27}$$

$$DisplayServer.outPort.RT = Pipe14.fromRole.RT \tag{28}$$

4 Bottleneck Analysis

Bottleneck analysis forms the core of system performance analysis. In our tool we have decided to apply bottleneck analysis laws of queueing networks. In the following we present these laws. A more detailed explanation can be found in [7]: The utilization of the i^{th} device(component or connector) is defined by:

$$U_i = X \times D_i \tag{29}$$

where X is the system throughput and D_i is the total service demand on the i^{th} device for all visits V_i of a task with processing time S_i. D_i is defined by the following law:

$$D_i = V_i \times S_i \tag{30}$$

where $V_i = X_i/X$, X_i is the i^{th} device throughput. Since utilization U_i cannot exceed 1, then $X \times D_i \leq 1$. Consequently, for each device i, the system throughput X checks the following law:

$$X \leq \frac{1}{D_i} \tag{31}$$

Therefore, the component or connector with largest D_i limits the system throughput and is the bottleneck. In other words, we must compute D_i for each device in the system. Let us assume that $D_j = Max\{D_1, D_2, ..., D_j, ..., D_n\}$; device j is the bottleneck.

In order to make these results into practice, our tool, automatically computes the demand property of each component and connector defined by the constituents properties:

$$D = \frac{(C.thruPut \times C.procTime)}{System.Throughput} \tag{32}$$

Then it picks up the device having the maximum value of D. There are several ways to deal with a bottleneck component: speed it up or reduce its demand in the system. After applying one of these options, we can re-compute the new quality attributes of the software and compare the improvements that will result. The main advantage of quality attributes analysis at architectural level is to detect such problems at an early stage. Since considering such changes at a late stage can be expense, difficult or even unfeasible.

5 An Automated Tool for Architecture Analysis

We have developed an automated tool that analyzes architectures according to the pattern discussed in this paper. This tool uses a compiler to map the architecture written in ACME+ onto Mathematica equations, then it invokes Mathematica to analyze and solve the resulting system of equations.

- We have defined an attribute grammar on top of ACME's syntax, which assigns attributes such as response time, throughput, failure probability to all the ports and all the roles of the architecture.
- We define semantic rules in the form of equations that involve these attributes and component/connector properties, and attach them to various BNF reductions of the syntax of ACME+.
- We have used compiler generation technology to generate a compiler for ACME+ language.

The tool takes as an input a file containing a given system architecture description written in our enriched ACME+ syntax. The compiler then translates this file into mathematical equations that characterize the system's non-functional attributes. Then, the tool invokes Mathematica to compute actual values of the system's attributes or to highlight functional dependencies between the attributes of the system and the attributes of the system's components and connectors. The equations are solved symbolically or numerically, depending on the goal of our analysis:

- Symbolically, by keeping component properties and connector properties unspecified, and having Mathematica produce an expression of the overall system attributes as a function of the component and connector properties.
- Numerically, by assigning actual values to component properties and connector properties and having Mathematica produce numerical values for the overall system.

In its current version, the compiler generates equations pertaining to response time, throughput and failure probability; each of these attributes corresponds to a tab in the GUI. Once we select a tab, we can perform the following operations:

- Compute the system level attribute as a function of component level properties. The GUI does so by merely solving the system of equations for the unknown $Sink.inPort.AT$, for attribute AT (where AT is the attribute identified by the selected tab). When a tab is selected, the GUI posts this value automatically.
- The GUI allows the user to update the value of a property of a component or connector, and will re-compute and post the updated value of the selected system level attribute.
- Once a tab is selected, the GUI also generates, and posts in a special purpose window, the symbolic expression of the corresponding attribute as a function of relevant properties of components and connectors.
- To enable a user to assess the sensitivity of the system level attribute with respect to component or connector level properties, the GUI shows a curve that plots the system level attribute on the Y axis and the component level property on the X axis.
- Finally, for some attributes [10], the GUI can also identify the component or connector that is the bottleneck of system performance for the selected attribute. Once the bottleneck of the architecture is identified, the user can change the value of its relevant property and check for the new (possibly distinct) bottleneck.

After analyzing the ACME+ description of Aegis system, the tool displays component and connector properties. It then invokes Mathematica in order to obtain symbolic and numeric values of system's properties and makes the results visible to the user. Based on the numeric results, the user may make modifications on component's or connector's properties and rerun the tool in order to obtain new system properties after changes. He repeats the process until an acceptable result is found. The performance analysis tool can be rerun as component's or connector's properties are modified, providing the user with incrementally improving feedback. The tool allows also the determination of component or connector bottleneck. In our example of Aegis system, $Trackserver$ component has the largest value demand and consequently, it is defined as the throughput bottleneck component. If the user wants to increase the throughput of the overall system, he can increase the throughput of $Trackserver$ component in order to maximize the overall impact. After introducing changes in the architecture, the user can re-compute the new quality attributes of the software and compare the improvements that will result. A demonstration of our tool can be downloaded from the following address: http://web.njit.edu/ mili/granada.exe.

6 Related Work

Over the past decade there has been considerable research devoted to modeling and analysis of software architectures. These approaches fall on two groups: qualitative and quantitative analysis.

6.1 Qualitative Analysis

Qualitative analysis approaches aims at making sure that the system has the expected properties, such as the lack of system crash, termination, reaching a specific state, etc.

In this context, several ADLs, like Darwin and PADL, relied on specific formal notation of process algebra to represent the architecture of a system and verify behavioral properties of its constituents. For example, Tracta [15] approach has been developed for the architecture described in the ADL Darwin. It describes the components in an algebraic notation FSP (Finite State Processes) process and allows the analysis of properties such as safety. Also, PADL [1,2] allows the analysis of deadlock, based on the classical process algebra.

6.2 Quantitative Analysis

Quantitative analysis aims to calculate the values of measurable properties of a system. All approaches have focused on the analysis of performance property and few others have chosen to analyze the reliability of systems. Several approaches have been proposed in this framework. Some have chosen to formally analyze systems at the architectural level, others process informally.

Formal Analysis. These approaches use mathematical proofs and methods for evaluating mainly operational quality attributes such as performance and reliability. They proceed in two steps: a modeling step aiming to the abstraction of a system to a model and an analysis step in which the genarated model is used to evaluate the desired quality attributes. Different formalisms were used for the modeling and evaluation of non-functional properties of a system: the Control Flow Graph (CFG), the Markov Process (MP), queuing networks (QN) and Stochastic Process algebra (SPA). These models can be used independently or they can be combined. Let's take the example of the tool proposed by Garlan et al. [27] which is based on Acme ADL and QN model. This tool transforms an architecture description written with Acme language to QN model. Then QN [6,14] equations are applied to estimate the system quality attributes.

Informal Analysis. Within the framework of informal analysis, several methods have been proposed for evaluating software architectures quality attributes. These methods can be categorized as either being experience-based, simulation-based, or scenario-based [23].

Experience-Based Evaluations. These approaches rely on the previous experience and domain knowledge of developers. Based on their previous experience, people who have encountered the requirements of the software system before can say if software architecture will be good enough. All experience-based approaches suffer from the deficiency that they rely on the presence of stakeholders and on a subjective analysis of quality attributes. Examples of methods in this group are ABAS [22], EBAE [24].

Scenario-Based Evaluations. These approaches try to evaluate a particular quality attribute by creating a scenario profile that forces a very concrete description of the quality requirement. A Scenario is brief description of a single interaction of a stakeholder with a system that expresses a particular quality attribute. Scenario-based evaluation methods require presence of relevant stakeholders to elicit scenarios according to their requirements. Important scenarios may be missed and therefore the evaluation fails to uncover risks and critical issues in software architectures. Examples of methods in this group are ATAM [24], SAAM [20], etc.

Simulation-Based Evaluations. These approaches rely on a high level implementation of some or all of the components in the software architecture. The simulation can then be used to evaluate quality requirements such as performance and correctness of the architecture. Examples of methods in this group are SAM [28], RARE/ARCADE [4]. Our work can be characterized by the following attributes, which set it apart from other work on architectural analysis.

- It is based on a relatively simple and generic architectural ontology,
- It is not restricted to only one quality attribute, but it proposes the analysis of many qualities: performance, reliability, availability and maintainability,
- It identifies bottleneck components or connectors and therefore helps in studying performance improvements,
- It is supported by an automated tool.

7 Conclusions and Future Work

In this paper, we discuss the need to develop an automated tool to analyze software architectures written in a formal ADL. Also, we propose ACME+ as an extension of ACME ADL, and discuss the development and operation of a compiler that compiles architectures written in this language to generate equations that characterize non functional attributes of software architectures. A demo of the tool that we developed, which includes the compiler and the user interface, is available online at: http://web.njit.edu/~mili/granada.exe. The work discussed in this paper is no more than a proof of concept to the effect that it is possible to reason automatically about non functional attributes of software architectures, given sufficient architectural information and component/ connector attributes. Our work focuses in pipes and filters architectural style and can be extended to other ones such as Client/Server which we are currently studying. In pipes and filters style, each component is characterized by only one functional dependency relation. We expect that analyzing systems in other architectural styles needs to define components with multiple functional dependency relation. Lets take the example of client/server style, if we consider the mini-architecture which contains three interacting components: a web server, a web client, and a database. We assume that the client makes requests to the server and receives responses from it. The server makes requests to the database in the process of filling the client's request. In ACME+ architectural description of the server component, we need to express two functional dependency relations. The first relation concerns the generation of a request to the database, the second one concerns the generation of a response to the client. Each relation describes an interaction between an input port, an output port and is characterized by a distinct quality attribute value. For example, the processing time of the first relation differs from the processing time of the second relation.

We believe that analyzing many other styles would be amenable to make the inductive rules more flexible and more generally applicable, by replacing the current inductive equations with inequalities, and replacing the current equation resolution by function optimization.

Dobrica and Niemela [8] suggest that the evaluation must be made against several quality attributes, which allows a better understanding of the weaknesses and strengths

of complex systems developed. We have also noted that most evaluation methods only address one quality attribute, and very few can evaluate several quality attributes simultaneously in the same method [23]. Concurrently, we have added to response time, throughput, failure probability, the following quality attributes: maintainability and availability and we envision to extend our work to analyse other non functional attributes.

References

1. Aldini, A., Bernardo, M.: On the usability of process algebra: An architectural view. Theoretical Computer Science 335(2-3), 281–329 (2005)
2. Aldini, A., Bernardo, M., Corradini, F.: A process Algebraic Approach to Software Architecture Design. Springer (2010)
3. Allen, R.J.: A formal approach to software architecture. Ph.D. Thesis, CMU (1997)
4. Barber, K.S., Graser, T., Holt, J.: Enabling iterative software architecture derivation using early non-functional property evaluation. In: Proc. 17th IEEE International Conference on Automated Software Engineering, pp. 23–27 (2002)
5. Bonta, E.: Automatic code generation: From process algebraic architectural descriptions to multithreaded java programs. Ph.D. in Computer Science University of Bologna, Padua (2008)
6. Buschmann, F., Henney, K., Schmidt, D.C.: Pattern-Oriented Software Architecture: On Patterns and Pattern Languages. John Wiley Sons (2007)
7. Denning, P., Buzen, J.: Operational Analysis of queueing network models. ACM Computer Surveys 10, 225–261 (1978)
8. Dobrica, L., Niemela, E.: A survey on software architecture analysis methods. IEEE Transactions on Software Engineering 28(7) (2002)
9. Frakes, W., Kang, K.: Software reuse research: Status and future. IEEE Transactions on Software Engineering 31(7) (2007)
10. Franks, G., Hubbard, A., Majumdar, S., Petriu, D., Rolia, J., Woodside, C.: A toolset for performance engineering and software design of clientserver systems. IEEE Transactions on Software Engineering 24(1-2), 117–136 (1995)
11. Garlan, D., Allen, R., Ockerbloom, J.: Exploiting style in architectural design environments. In: Proceedings of SIGSOFT 1994: Foundations of Software Engineering (1994)
12. Garlan, D., Monroe, R.T., Wile, D.: Acme: An architecture description interchange language. In: CASCON 1997, Toronto, Ontario, pp. 169–183 (1997)
13. Garlan, D., Schmerl, B.: Architecture-driven modelling and analysis. In: SCS 2006 Proceedings of the Eleventh Australian Workshop on Safety Critical Systems and Software, p. 69 (2006)
14. Garlan, D., Shaw, M.: An introduction to software architecture: Perspectives on an emerging discipline. Prentice Hall (1996)
15. Giannakopoulou, D., Kramer, J., Cheung, S.C.: Behaviour analysis of distributed systems using the tracta approach. In: Proc. 17th IEEE International Conference on Automated Software Engineering, p. 735 (1999)
16. Kazman, R., Bass, L., Abowd, G., Webb, M.: Saam: A method for analyzing the properties of software architectures. In: Proc. 16th International Conference of Software Engineering (1994)
17. Klein, M.H., Kazman, R., Bass, L., Carriere, J., Barbacci, M., Lipson, H.: Attribute-based architecture styles. In: Proc. TC2 First Working IFIP Conference on Software Architecture (WICSA1), pp. 225–244 (1999)

18. Bass, L., Clements, P., Software Architecture, R.K.: Software Architecture in Practice. Addison-Wesley (2003)
19. Luckham, D.C., Kenney, J.J., Augustin, L.M., Vera, J., Bryan, D., Mann, W.: Specification and analysis of system architecture using rapide. IEEE Trans. Software Eng. (2000)
20. Maurya, L.S., Hora, H.: Comparison of software architecture evaluation methods for software quality attributes. Journal of Global Research in Computer Science 1(4) (2010)
21. Medvidovic, N., Oreizy, P., Robbins, J.E., Taylor, R.N.: Using object-orlenfcd typing to support architectural design in the c2 style. In: Proceedings of ACM SlGSOFT 1996. Fourth Symposium on the Foundations of Software Engineering (FSE4), pp. 24–32 (October 1996)
22. Moriconi, M., Qian, X.: Riemenschneider, R. A.: Correct architecture refinement. IEEE Transactions on Sofrware Engineering, 356–372 (April 1995)
23. Maurya, L.S., Hora, H.: Comparison of software architecture evaluation methods for software quality attributes. Journal of Global Research in Computer Science 4(1) (November 2010)
24. Balsamo, S., Bernardo, M., Simeoni, M.: Performance evaluation at the software architecture level. In: Bernardo, M., Inverardi, P. (eds.) SFM 2003. LNCS, vol. 2804, pp. 207–258. Springer, Heidelberg (2003)
25. Schmerl, B., Garlan, D.: Acmestudio: Supporting style centered architecture development. In: Proceedings of the 26th International Conference on Software Engineering (2004)
26. Shaw, M., DeLine, R., Klein, D.V., Ross, T.L., Young, D.M., Zclesnik, G.: Abstractions for software architecture and tools to support them. IEEE Transactions on Software Engineering, 314–335 (April 1995)
27. Spitznagel, B., Garlan, D.: Architecture-based performance analysis. In: Proceedings of the Conference on Software Engineering and Knowledge Engineering, CA (1998)
28. Wang, J., He, X., Deng, Y.: Introducing software architecture specification and analysis in sam through an example. Information and Software Technology, 451–467 (May 1999)

An Experiment on Self-configuring Database Queries

Pietu Pohjalainen

Department of Computer Science,University of Helsinki, Finland
pietu.pohjalainen@cs.helsinki.fi
http://www.cs.helsinki.fi/

Abstract. Database access is in the core of many software systems. When building an object-oriented system with a relational backend, a common approach is to use an object-to-relational mapping component to make the relational database oblivious to the application programmer.

Self-configuring database queries is a way to reduce the effect of internal dependencies within a software. To assess its usability, we organized a randomized, controlled study for software engineering students, who were given a number of maintenance tasks on a sample software with two versions: the first one using transparent persistency as a control group and the second using self-configuring queries.

Although the attendees in both groups used equal time in completing the tasks, it turned out that the group using self-configuring queries outperformed the control group in code quality by a factor of three. This gives us evidence to believe that self-configuration in a software's architecture can be beneficial for maintainability.

Keywords: Relational Databases, Self-configuring Software Architecture, Randomized Experiment, Controlled Experiment.

1 Introduction

Relational databases are used for storing data in virtually all kinds of information systems. Telephone switching systems, e-commerce transaction systems, human resources information systems and e-mail indexing systems, for example, have implemented data persistency by relying on relational database engines. Although these engines often include some support for building application logic into them, it is popular to build applications using a general purpose language, such as Java or C#.

In these cases the way in which the relational database is used is a major architectural decision. Using plain database interfaces, such as Java's database connectivity interface [1] easily leads to tedious work of manually implementing routines for storing and retrieving data to and from the database. Some researchers estimate that the low-level code for accessing a database can be up to 30% of the total line count of a normal business application written in direct access style [2,3]. Manual labour is known to be error-prone and have low productivity figures. For this reason, many architects choose to use object-to-relational mapping components, such as those defined by the Java persistency interface [4] to handle the mismatch between object-oriented program logic and relational database models.

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 33–47, 2013.

Object-to-relational mapping (ORM) components aim to reduce the manual labour needed in building object-oriented database applications. Instead of manually defining database queries for retrieving application data, an external component handles the generation of the needed queries. This is done by defining mappings between application objects and database tables. These mappings define how object-oriented structures, such as inheritance, object collections, and other object links should be made persistent in the database. The goal of this component is to liberate the object-oriented programmer from thinking at a relational database abstraction level. For this, the we use the term *transparent persistency*.

Transparent persistency refers to the idea of data persistency should not affect the way programmers build the application logic. Instead, persistency should be a modular aspect, as envisioned in aspect-oriented programming [5]. At first glance, this seems to be a clever idea: in many cases, the routines needed for storing and retrieving data from the database makes algorithms look unnecessarily complex and cluttered.

However, the drawback of the approach is that the exact workings of the software system are blurred, since the database accessing code is faded away from the working view of the programmer. Since transparent persistency removes the link between algorithms using the data from the database and the accessing code, programmers who are not intimately familiar with the system may conceive an incorrect understanding of the system.

Self-configuring components are a recently introduced novel approach to building links between different software layers. The idea is to recognize automatically resolvable dependencies within the codebase, and to implement corresponding resolver components. This helps in maintenance, since instead of the programmer manually following all the dependencies in the event of a functionality change, the automated component does the task. Examples have been implemented in areas of building consistent user interfaces [6] and database queries [7].

Currently, it is not very well known how these architectural decision affect programmer performance. To gain understanding on what works in practice, empirical validation is needed. To better understand the effect of transparent persistency's consequence for maintenance, we have formulated the following research questions regarding development speed and quality:

An initial hypothesis for introducing configuring component that is based on meta-programming on byte-code level would suggest that at least initially it would be harder and slower to produce correctly working code. Probably after a run-in period, the programmers involved would learn to understand the workings and responsibilities of the meta-programming component. This leads us to formulate a null hypothesis:

RQ_1: Do self-configuring components make it faster to perform maintenance tasks?

RQ_2: Do self-configuring components produce better quality in maintenance tasks?

RQ_3: Do self-configuring components make programmers more productive?

To gain initial insight in these questions, we conducted a randomized, controlled experiment on using Java persistency interface for database queries. In this experiment, the attending students were randomly divided into two groups: the transparent persistency group as the control group and the self-configuring database queries as the experimental treatment group. Attendees were given slightly modified versions of the same software

and were asked to perform given maintenance tasks upon the software. We tracked the time for performing the asked modifications. Each submission was afterwards graded for correctness. Productivity is defined as a ratio of the number of correct submissions per hour used on the tasks.

The rest of the paper is organized as follows. Section 2 gives a short introduction to self-configuring software components, object-to-relational mappings and implementation techniques used in the presented experiment. Section 3 defines the used research methodology. Section 4 shows results obtained in the experiment. Section 5 discusses these findings. Section 6 concludes the paper and outlines future work.

2 Preliminaries

In this section we first review the object-to-relational approach to using databases in object-oriented programs. We then introduce self-configuring queries in the context of the example used in the experiment. Object-to-relational mapping components are commonly used to allow developers concentrate to code within one paradigm. The idea is that special mappings are used to translate an object's persistent fields into corresponding relational database concepts.

These mappings provide a placeholder for defining handling rules for issues rising from the mismatch between object and relational paradigms. Examples of these problems are e.g. the problem of handling inheritance, navigability of one-way object links versus bi-directionality of relational links, and representation of object collections, to name a few. The mapping also provides a place for defining the fetching rules for certain object classes. This is often a major source of performance problems in applications using the object-to-relational approach for persistency.

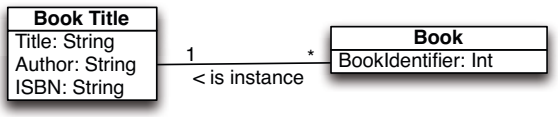


Fig. 1. One-to-many mapping in the experiment application

The fundamental problem in using default rules in database fetches is that most often the set of data objects being fetched from the database is context sensitive. For example, let us consider a fragment of our experiment's class diagram that is presented in Figure 1. The model shows a one-to-many connection between a book title and a number of books. The idea is to show that for each book title, its author name and ISBN number are stored only once; and for each book copy, there's a distinct object in the system.

With this simple example, we can see two use cases with different database queries. These cases are implemented as shown in Table 1: the first one lists all book titles and fetches only objects of the *BookTitle* class. The second also lists all book copies and thus touches objects of both classes in the system.

Given these two use cases, using some fixed fetching rule for the BookTitle class is always somehow wrong: if the default rule is to eagerly fetch Books with its BookTitle,

Table 1. Algorithms for listing all book titles and book copies

```
procedure PRINTTITLES(List allTitles)
    for all title in allTitles do
        print title.Author
        print ": "
        println title.Title
    end for
end procedure
```

```
procedure PRINTBOOKS(List allTitles)
    for all title in allTitles do
        print title.Author
        print ": "
        println title.Title
        cnt ← 0
        for all book in title.Books do
            println book.BookIdentifier
            cnt ← cnt + 1
        end for
        println cnt + " copies."
    end for
end procedure
```

then the execution of *PrintTitles* method is unnecessarily slow, since the algorithm does not need all that information. However, if the default rule is not to fetch associated books with their titles, then method *PrintBooks* cannot be executed unless the default rule is overridden or some other internal magic is performed.

A common approach in this case is to use the Proxy design pattern [8] to guard against cases where the algorithm is requiring some object data that is not fetched from the database. The proxy represents the object's external interface, and upon usage of the object data, it generates the necessary database queries needed to lazily retrieve that data from the database [9].

This is a way to implement transparent persistency: all algorithms can be written in an object-oriented language, and it is the responsibility of the object-to-relational mapping component to provide the implementation for relational database queries.

The downside of this approach is that it is easy to implement inefficient algorithms, as the proxying approach queries the database only at when a certain object is first accessed. This creates the *n+1 queries problem*, which happens when traversing collections of objects. The first query to the database fetches the root object. For each object link in the root object that is being accessed by the traversal algorithm, a new query is generated. Due to $n + 1$ round-trip queries to the database, performance usually severely degrades.

For this reason, the object-to-relational mapping frameworks provide means to explicitly specify a query for fetching an initial data set. However, this approach is arguably an inelegant solution, since it breaks the promise of transparent persistency and generates dependencies between the explicitly specified query sites and actual data usage sites.

One solution to provide more precise control over the fetched data set is to use self-configuring database queries [7]. In this approach, the software contains a query-resolved component, which analyzes the code that accesses the database and estimates the right database query to use. It is claimed that the component improves software maintainability, since the database usage is made more explicit, yet the dependencies between the data accessing algorithms and data fetching code are automatically resolved.

To the application programmer, the use of self-configuring database queries is shown by the need to parametrize database access with the algorithm that the queried data is

being subjected upon. In traditional transparent persistency, the programmer fetches an initial object, e.g. a certain *Title* from the database and then executes the handling code, e.g one of the methods in Table 1. With self-configuring queries, instead of providing an initial guess for the needed data, the programmer gives the name of the algorithm (*PrintBooks* in this case) to the self-configuring component. The self-configuring component analyzes the behavior of the algorithm and provides an approximation of the required data for running the algorithm with an optimal number of database queries.

In our implementation, the self-configuring query component analyzes the byte code of the method in the containing Java class. The configurator builds a control-flow graph of method execution, and includes database joins to the generated query whenever a class with object-to-relational mapping is encountered in the control flow. When the self-configurator component is given the code shown in Table 1, it deduces that a query set of {book titles, booktitle.books} is needed for executing the method *PrintBooks* while a query set of {book titles} suffices for execution of the method *PrintTitles*.

This approach is believed to give benefits in two ways: improved maintainability and improved modularity. Maintainability is improved, because the database query site does not explicitly define what data to fetch from the database, but instead the self-configuring query determines that. For this reason, the database accessing algorithms can be changed more easily, since dependence on the database query site is automatically updated to reflect the updated algorithm.

Modularity is improved because the one self-configuring component can handle a number of different usage sites. This is an improvement of the traditional n-tiered software architecture, where the database access layer needs to contain functionality specific to business logic. In the context of this experiment, the traditional way would be to implement one database query that fetches all book titles for executing method *PrintTitles* and another query that joins all the book titles with the books for running method *PrintBooks*.

3 Methodology

To gain understanding on how this approach works in practice, we recruited a sample of 16 students to attend a randomized, controlled experiment. About half of the students were freshmen (n=10), and the rest were in their final bachelor year or master's students (n=6). Before attending the test, a two-hour lecture on concepts of relational databases, object-oriented programming and object-to-relational mapping tools was given to each participant.

We built two versions of a simple library book-keeping application. The application consists of five classes that are programmed to manage book titles, books, lenders and book loans in a database. Each attendee is randomly assigned to one of the versions, in which he stays during the whole experiment. The baseline application contains a functionally working version of the software with the same set of functionality implemented in both of them. For example, both methods listed in Table 1 are contained in the package given to each test subject.

The first version (group 'transparent persistency, TP') of the sample application was written in the style of transparent persistency: the complexity of handling database

queries is hidden behind the persistency framework's internals, which in this case was to use bytecode-instrumented proxying implementation of database queries. In the second version (group 'self-configured, SC'), the application contained method calls to self-configuring instructions to automatically prefetch the correct objects from the database, and to rewrite the corresponding fetching query.

In the variant given to the self-configured group, the database access is not fully obscured away. The database access component is parametrized with the actual algorithm that is going to be executed. The component analyzes the algorithm and produces the database query to be used at the subsequent database access time. This construct helps to remove dependencies between database access code and the actual algorithms that are being executed. A change in the algorithm is automatically reflected by the self-configuring component, thus removing the need to manually update the database query.

The initial application consists of a functionally consistent set of program code. The test subjects are asked to perform modifications to the sample application; depending on the skill and speed of the attendee, up to 7 functional tasks can be performed during the trial. Each task was time-boxed, meaning that the attendee is asked to return his version even if it is incomplete when the time is up. If an attendee thinks that he/she cannot be performing any more tasks, he is free to leave at any given point. Attendees were awarded with credit units or extra points to a course test regardless of how many tasks they opted to complete during the test. All test subjects were physically present in the classroom during the experiment. The submission mechanism was to send the source code for each application version via e-mail after each completed task.

To moderate the application complexity, the sample application was written as a command-loop tool. In the following code listings we will illustrate the differences between the two groups' variants using the following notation: line prefix "TP" notifies that this line is present only in the transparent persistency variant. "SC" lines are present in the self-configured variant. "C" lines are common to both of the variants.

Using this notation, the command line loop for the use case of listing all book copies in the database is implemented as listed in Program 1.

Program 1. Two variants of the *list books* use case

```
C: switch(cmd) {
C: [..]
C:   case "list books":
SC:    String alg="BookTitles.PrintBooks";
SC:    Resolver r = new Resolver(alg);
SC:    bookTitles.fetchFromDatabase(r);
TP:    bookTitles.fetchFromDatabase();
C:     bookTitles.printBooks();
C:     break;
C: }
```

As can be seen in the listing, the only difference between the two variants is that the self-configured version contains a stronger hint of the fact that this command is accessing the database. The *Resolver* component is parametrized with the name of the

algorithm that is going to be executed subsequently. It is important to note that in the transparent variant, although the name of the method suggests that the method is going to access the database, there is no explicit link between the data-fetching code and the algorithm used to print out the books. Consequences of this omission are further explored later in the results section.

In the resolver version, the component analyzes the internal workings of the method that is given as an argument and produces an estimate of what data objects should be fetched. This gives the maintainer a stronger binding between the data-fetching code and the algorithm used to process the fetched data.

The fetchFromDatabase routine as listed in Program 2 also differs a bit in the two variants. The variant for the transparent persistency group creates a database query by using the object querying language defined in the Java persistency interface. This query fetches an initial set of book titles from the database. In this variant when the execution arrives to a book copy that has not yet been fetched from the database, the automatically generated proxy instance generates a new query to get the missing information.

Program 2. Two variants of database fetching code

```
SC: void fetchFromDatabase(Resolver r) {
TP: void fetchFromDatabase() {
TP:  Query q;
TP:  q = createQuery("from BookTitle");
SC:  Criteria q = r.resolve();
C:
C:   bookTitles.clear();
C:   List<BookTitle> list = q.list();
C:   for(BookTitle bt : list)
C:       bookTitles.add(bt);
C: }
```

In the self-configuring variant, the method name implementing the command is given as an argument to the resolver component. The resolver analyzes the code of the algorithm and initiates a database query to fetch the required data. If the resolver happens to underestimate the required set of objects, the normal proxy-based fallback is used as a safeguard. At this point, it is important to note that the self-configured version does not contain the matching between database and business logic, but instead it operates in the business logic domain.

The essential difference between the two variants is the degree of how explicit the database query is. In the transparent persistency variant, the database access code is minimized, with the design goal of reducing the mental load of the programmer, as he should not be worrying about how to access a database. This can also be misleading, since there is no hint of the mechanism of how the persistency framework should be used. In the control group variant, the database accessing is more explicit, since the algorithm calling site generates the database query component before actually executing the query.

Armed with the initial example cases shown in Table 1 given to the test subjects, they are asked to implement a number of maintenance tasks to the software. All of the

tasks follow the same pattern: fetch some data from the database and then print out information. Task 0, which was to change the source code character set to UTF-8, was not graded. It was used to give the attendees an opportunity to gain understanding of the sample application and to practice the submission procedure. The tasks are summarized below:

Identifier	Task description
Task 0	Fix source code charset (not graded, used to practice the submission procedure)
Task 1	List all book loaners in the system
Task 2	List all book loans in the system
Task 3	Fetch specific book title by its ISBN
Task 4	Modify task 3 result to print out all books associated with the given title
Task 5	Fetch specific lender by his social security number
Task 6	Modify task 5 result to print out all books borrowed by this lender
Task 7	Modify the list of book titles to print out all lent books associated with that title

The tasks list contains maintenance tasks that can be classified as adaptive maintenance in IEEE Std 14764-2006 terms [10]. Each of the tasks delivers an added functionality that is requested to make the sample software able to be a better fit for its task of handling a database of library books. Each of the tasks are implementable without inducing new requirements or other needs for architectural changes.

4 Results

We graded each submission based on correctness on a dichotomous scale: a correct submission to a given task is programmed to connect to the database, fetch certain data, and print out corresponding results. Minor stylistic problems, such as output formatting did not affect the grading.

In the freshmen group, the assignments were generally considered to be hard. This is understandable, since outside the persistency code, the implementation followed object-oriented principles in both versions. However, even in this group, some test subjects succeeded in getting some of the tasks submitted correctly.

The second problem to occur frequently among all test subjects was the failure to fetch the database correctly. Often the modified application seemed to work perfectly, but an underlying problem (”a bug”) was introduced by accidentally re-using the applications internal data instead of targeting the query to the database. This problem was most often seen in the transparent persistency group. We believe this to be the consequence of the interface between business logic and database access code. Since the business logic explicitly defines only the initial object of the processed object graph, thus making all other database access translucent, the students often failed to correctly handle the cached versions of the objects. Other programming errors included various incorrect structures, such as null pointer exceptions, cache-handling errors etc. Each of these were unique to the given submission.

For each test subject, we noted the state of his studies (freshman/advanced), randomization group (transparent persistency / self-configured), time spent on each submission (rounded to next 5 minutes) and the correctness of the submission.

To study the productivity between the two approaches, we measured the time for producing a submission to all given tasks. However, we are analysing this information in the advanced group only. The test subjects in the freshmen group are omitted due to their submissions having such low success rate; there is only one pair of submissions where two test subjects belonging to different randomization groups have been able to produce a correct submission to the corresponding task (test subjects #7 and #8 for task 3).

In the advanced students' group, there was one case, where the test subject had to leave early due to a medical condition before he was able to send a single submission. He was randomized to the self-configuring. Due to the nature of that situation, his results are not included in the analysis. Outside this case, no test subject left the experiment prematurely.

Subject #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Study state	FM	FM	FM	FM	FM	FM	FM	FM	FM	FM	MS	MS	MS	MS	MS	MS
Randomization	SC	TP	TP	SC	SC	TP	SC	TP	SC	TP	SC	SC	TP	SC	TP	TP
Success rate	0/2	0/2	0/1	1/2	1/3	0/3	2/3	1/3	0/3	0/3	7/7	5/7	4/6	7/7	0/5	0/3

Fig. 2. Summary of the results

Figure 2 summarizes the experiment results. Study state refers either to a freshman (FM) or master's student (MS). Randomization group is either transparent persistency (TP) or self-configuring queries (SC). In success rate (x/y) x marks the number of correct submissions and y measures the number of attempted tasks.

4.1 RQ1

Given this information, we can return to our research questions. RQ1 proposes that self-configuring queries make it faster to perform maintenance tasks. According to Figure 2, the test subjects in the transparent persistency group returned a total number of 27 submissions, be they correct or incorrect. The test subjects in the self-configured group returned 33 submissions. There are an equal number of subjects in each of the randomization groups, and the maximum allowed time was the same for everybody.

The self-configuring turned in more submissions, but the difference between the two groups is small. From this viewpoint, we cannot support the idea of transparent persistency making the completion of these maintenance tasks faster, but the self-configured group does not get a decisive victory, either.

4.2 RQ2

Figure 3 summarizes the frequency of correct submissions. For example, two attendees in the self-configured group got zero submissions correct and six attendees in the transparent persistency group got zero submission correct.

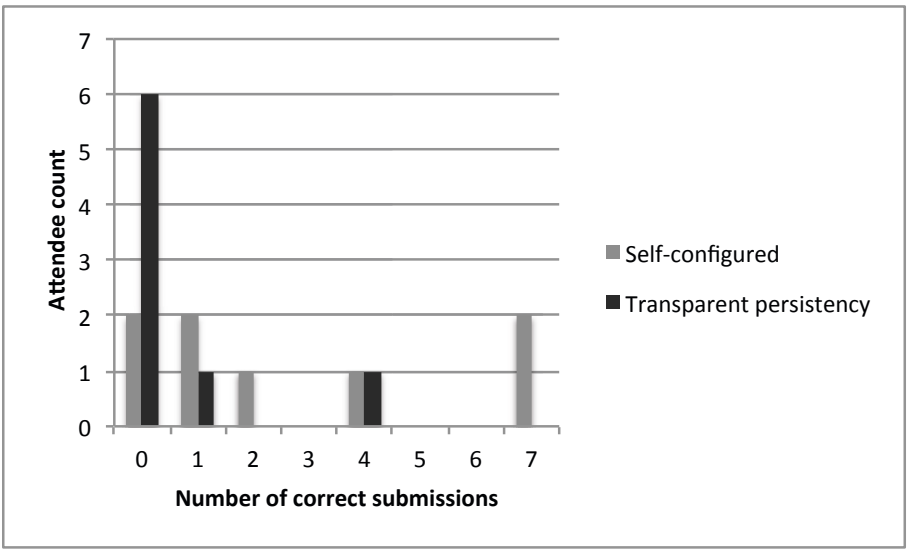


Fig. 3. Count of correct submissions for both groups

The grading distribution in Figure 3 leads us to formulate the null hypothesis for RQ2:

> H_0: Transparent persistency makes it easier to produce correct database handling code than self-configuring queries.

When the test data is normally distributed, a parametrized test should be used to determine differences between populations. Otherwise, a non-parametric test should be used [11, p. 450-452]. Our data is based on human behavior, and could be believed to be normally distributed. However, the results look like a non-normal distribution. For this reason, we calculated both parametric and non-parametric statistical tests.

Both the one-tailed independent samples t-test and Mann-Whitney U test result a p-value of $\alpha < 0.05$, giving a suggestion that the null hypothesis needs to be rejected. Informally, the self-configured group was performing much better than the transparent persistency group: two attendees who got assigned the self-configuring, query rewriting-based base software got all seven graded tasks implemented correctly. On the other hand, six attendees using the traditional transparent persistency failed to produce a single correct submission.

4.3 RQ3

Finally, as a measure of productivity, we consider the time consumed in programming tasks in the non-freshmen group. The times for producing a *correct* submission (n=23) varied between 5 and 55 minutes. There was no statistically significant difference in time consumed between the two groups.

As the number of correctly returned submission series is low, statistical analysis of these series would not serve a purpose. As a reference information, when ordering the correct by-task submissions as a speed contest, all test subjects in the self-configured group were producing submissions faster, on average.

Productivity was defined as the number of correct submissions per time consumed. Thus, for each individual we calculate his productivity index as $\frac{\sum_{i=1}^{7} time_i}{\sum_{i=1}^{7} corr_i}$ where the variable $time_i$ refers to the time for producing the answer for task i and $corr_i$ is 1 for a correct answer and 0 for an incorrect answer in task i. This formula slightly favors the productivity of incorrect answers, since the time needed to rework an incorrectly piece of software, as would happen in a real-world software development situation, is not included in this productivity index.

Applying this formula to the test subjects in the advanced group yields a productivity index of 105 minutes per correct task submission in the transparent persistency group. For the self-configured group, the index stands at 24 minutes.

For producing correctly working code, the self-configured group outperformed the transparent persistency group by a factor of four.

4.4 Threats to Validity

We have found a statistically significant effect between two alternative ways of querying a database in an object-oriented program. However, a number of threats to the validity of this finding do exist.

In general, a controlled experiment like this differentiates from industrial use in the scope of time that can be spent in understanding the associated technologies. In an industrial setting, programmers usually work with the same technology stack for months or years. In this study, some of the test subjects were exposed to the technologies for the first time. However, some of this concern can be argued to be mitigated by the randomization process; these technologies can be assumed to be equally unknown to both groups. A related problem is the small size of the software used in the experiment. It might be the case that the complexity of transparent persistency pays off only after a certain software size has been achieved.

Another concern is that using students in controlled experiments limits the generalizability of results to a industrial setting. However, previous studies, e.g. [12] have found a high correspondence between (1) undergraduate and junior programmers; and (2) graduate and senior programmers in terms of understanding object-oriented programming, which can be interpreted to mean that at least half of the experiment population were up to industrial standards. This distribution of attendees is probably not fully unrealistic, since companies often use junior programmers to perform maintenance tasks [13]. Another interpretation of our work is that the finding in [12] is probably valid: the advanced group in our study was able to grasp the functionality of the object-oriented system, while the freshmen group clearly had problems understanding it.

Our third concern is the the sample size, which is small. We conducted this study on 17 students, of whom 10 were in their first year, and 7 students were more advanced. Since one student in the advanced group was excluded from the experiment due to a medical condition, the total number of test subjects is 16. Due to the small sample size, individual performance shows a large role in our statistical analysis.

Our fourth concern regards the use of number of submissions as productivity measure. Most of the tasks are equidistant in substance in the sense that for most tasks, the task n is not dependent on whether he succeeds in $n+1$. For tasks $T4$ and $T6$ there is a dependency to the previous task. This dependency plays some role for subjects #12 and #13. Subject #12 failed in task T3, but was able to fix the problem in T4 submission. He also got task T5 correct but failed his task T6. The latter dependency is present for subject #13 as well.

5 Discussion and Related Work

The use of transparent persistency for storing objects is not a new idea. A number of systems for various languages and programming environments have been described in the literature; [2,14,3], to name a few.

Similarly to the self-configured queries, researchers have used the program source as a source model for model transformations in various contexts. For example when building portable interpreters, the opcode definitions can be implemented in the interpreter implementation language. When the definition is compiled, the end result, in byte code or other low-level representation can be used as a building block when interpreting the opcode in question [15,16].

Using program analysis for extracting database queries has been proposed by various researchers. We used the component documented in previous work [7], using the same object-to-relational mapping framework, Hibernate [3]. In addition to this style, Ibrahim and Cook have proposed a dynamically optimizing query extractor [17]. Wiedermann et. al have implemented a version which combines static and dynamic techniques [18]. In the simple examples used in this experiment, a simple analyzer was able to deduce the required queries. However, when the business logic involves more complex decisions, the automated self-configuring component needs to be updated to be able to handle the more complex case. In a way, the situation resembles the full-employment theorem of compiler writers [19]: since the business logic can be arbitrarily complex, the evolution for implementations of automated resolvers are restricted by laws of economics rather than some self-configurator being optimal to every case.

Although various systems for implementing transparent persistency have been available for decades, it seems that its actual usage patterns have not been studied very well. This implementation was initially employed in an industrial project for building a product-line for provisioning governmental mobile subscriptions [20], where the need for modular constructs was critical to development success and transparent persistency turned out to lack the support for efficient modularity. Piccioni et al. have studied the usability of their database access library by use of structured interviews and controlled experiments [21]. However, according to our best knowledge, comparisons of the effect of transparent persistency versus any alternative have not been empirically studied so

far. Considering the fact that transparent persistency is a frequently employed technique in industrial software engineering, the lack of empirical studies is a surprise.

The use of orthogonal, transparent persistency can be seen as a specialized case of modularizing orthogonal features. To relate our findings, we can refer to literature on empirical studies of aspect-oriented programming [5]. We can consider transparent persistency to be one of the cross-cutting aspects of the software.

The empirical response to productivity in aspect-oriented programming seems to be mixed. For example, in [22], the researchers were unable to find any positive impact of using aspect orientation for building maintainable and easier-to-comprehend software.

Kulesza et al. studied the use of aspect-oriented programming for a maintenance task [23]. They implemented two similar systems, the first one in an object-oriented fashion and the second one in an aspect-oriented fashion and compared both systems' internal metrics, such as size, coupling and cohesion, before and after a specified maintenance task. Their conclusion is that the aspect-oriented style was superior in terms of observed software characteristics. They did not measure the time spent on producing the different versions of the software, nor the time spent on implementing the maintenance tasks. Thus, productivity of the maintenance task cannot be assessed. Contrary to our study, the person doing the maintenance task was presumably an expert programmer and familiar with the system being maintained. In our setting we had a sample of junior programmers performing a series of maintenance tasks on a previously unknown system.

Endrikat and Hanenberg measured the time for building a system and then performing a number of adaptive maintenance tasks on a number of test subject [24]. For many of the tasks, the object-oriented way in the control group was faster, but for combined build+maintenance time they suggest that aspect orientation can be beneficial.

Another, indirect finding on this study was the poor programming performance in the freshmen group: most of test subjects in this group were having serious problems with the experiment. Although they had completed the first programming courses offered by the university and were given a fully functioning system, they failed to perform even simplest modifications to it. For this reason the university has already adjusted our first programming courses to use a new studying method based on extreme apprenticeship [25]. In future work, it would be interesting to compare how these adjustments to teaching methodologies have affected freshmen programming skills in maintenance tasks.

6 Conclusions

We performed a randomized, controlled experiment to assess the usefulness of transparent persistency. In the experiment we built two versions of a small sample application and asked a number of test subjects to perform a number of maintenance tasks upon their software variant.

We measured the time used to do these maintenance tasks and related them to successful submission rates. As a result, we concluded that the self-configured group was performing much better in terms of producing correctly behaving code.

In terms of correctly returned submissions, the self-configured group outperformed the transparent persistency group by a factor of three. This result is a statistically significant difference (p value $\alpha < 0.05$) between the two populations.

In productivity, the self-configured group outperformed the transparent persistency group by a factor of four: on average, the self-configured group was able to produce a correct submission in under half an hour. In the transparent persistency, on average it took almost two hours to do the same. However, due to the low sample size, we did not perform any statistical analysis on productivity.

This result casts a shadow of disbelief on the concept of transparent persistency: if the application is going to access a database, it probably is not a good idea to try to disguise its functionality to be something else. This disguise seemed to be the main source of program miscomprehension within the test subjects in the transparent persistency group. However, the small sample application limits the generalizability of the result. Experiments with larger software and larger populations are needed to understand the usefulness of transparent persistency for software development. On the other hand, the result gives an initial empirical validation of the usefulness of using self-configuring software components to reduce the maintenance effort and to improve architectural modularity.

References

1. Fisher, M., Ellis, J., Bruce, J.C.: JDBC API Tutorial and Reference, 3rd edn. Pearson Education (2003)
2. Atkinson, M.P., Bailey, P.J., Chisholm, K., Cockshott, W.P., Morrison, R.: An approach to persistent programming. Comput. J. 26, 360–365 (1983)
3. Bauer, C., King, G.: Hibernate in Action (In Action series). Manning Publications Co., Greenwich (2004)
4. DeMichiel, L., Keith, M.: JSR 220: Enterprise JavaBeans 3.0. Technical report, Sun Microsystems (2007)
5. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., 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)
6. Pohjalainen, P.: Self-configuring user interface components. In: Proceedings of the 1st International Workshop on Semantic Models for Adaptive Interactive Systems, SEMAIS 2010, pp. 33–37. ACM, New York (2010)
7. Pohjalainen, P., Taina, J.: Self-configuring object-to-relational mapping queries. In: Proceedings of the 6th International Symposium on Principles and Practice of Programming in Java, PPPJ 2008, pp. 53–59. ACM, New York (2008)
8. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
9. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
10. International Standard - ISO/IEC 14764 IEEE Std 14764-2006 software engineering #2013; software life cycle processes #2013; maintenance. ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998, pp. 1–46 (2006)
11. Robson, C.: Real World Research: A Resource for Users of Social Research Methods in Applied Settings. John Wiley & Sons (2011)

12. Arisholm, E., Sjoberg, D.I.K.: Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. IEEE Trans. Softw. Eng. 30, 521–534 (2004)
13. Gorla, N.: Techniques for application software maintenance. Inf. Softw. Technol. 33, 65–73 (1991)
14. Atkinson, M., Morrison, R.: Orthogonally persistent object systems. The VLDB Journal 4, 319–402 (1995)
15. Elliott, C., Finne, S., de Moor, O.: Compiling embedded languages. Journal of Functional Programming 13 (2003)
16. Yermolovich, A., Gal, A., Franz, M.: Portable execution of legacy binaries on the Java virtual machine. In: Proceedings of the 6th International Symposium on Principles and Practice of Programming in Java, PPPJ 2008, pp. 63–72. ACM, New York (2008)
17. Ibrahim, A., Cook, W.R.: Automatic prefetching by traversal profiling in object persistence architectures. In: Thomas, D. (ed.) ECOOP 2006. LNCS, vol. 4067, pp. 50–73. Springer, Heidelberg (2006)
18. Wiedermann, B., Ibrahim, A., Cook, W.R.: Interprocedural query extraction for transparent persistence. In: Proceedings of the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications, OOPSLA 2008, pp. 19–36. ACM, New York (2008)
19. Rice, H.G.: Classes of recursively enumerable sets and their decision problems. Trans. Amer. Math. Soc. 74, 358–366 (1953)
20. Pohjalainen, P.: Bottom-up modeling for a software product line: An experience report on agile modeling of governmental mobile networks. In: Proceedings of 15th International Software Product Line Conference, SPLC 2011, pp. 323–332 (2011)
21. Piccioni, M., Furia, C.A., Meyer, B.: An empirical study of api usability. In: Proceedings of 7th International Symposium on Empirical Software Engineering and Measurement, ESEM 2013 (2013)
22. Bartsch, M., Harrison, R.: An exploratory study of the effect of aspect-oriented programming on maintainability. Software Quality Control 16, 23–44 (2008)
23. Kulesza, U., Sant'Anna, C., Garcia, A., Coelho, R., von Staa, A., Lucena, C.: Quantifying the effects of aspect-oriented programming: A maintenance study. In: Proceedings of 22nd IEEE International Conference on Software Maintenance, ICSM 2006, pp. 223–233 (2006)
24. Endrikat, S., Hanenberg, S.: Is aspect-oriented programming a rewarding investment into future code changes? A socio-technical study on development and maintenance time. In: Proceedings of IEEE 19th International Conference on Program Comprehension, ICPC 2011, pp. 51–60 (2011)
25. Vihavainen, A., Luukkainen, M.: Results from a three-year transition to the extreme apprenticeship method. In: 2013 IEEE 13th International Conference on Advanced Learning Technologies (ICALT), pp. 336–340 (2013)

Automated COSMIC-Based Analysis and Consistency Verification of UML Activity and Component Diagrams

Asma Sellami[1], Mariem Haoues[1], and Hanêne Ben-Abdallah[2]

[1] University of Sfax, Mir@cl Laboratory, Tunisia
[2] FCIT, King Abdulaziz University, K.S.A.
asma.sellami@isimsf.rnu.tn, mariem_haoues@yahoo.fr,
hbenabdallah@kau.edu.sa

Abstract. UML has been established as a de facto standard for modeling software. It offers a set of complementary diagram types used to document functional, dynamic and static views of a system. UML diagrams diversification and their multi-view representation can cause inconsistencies among the diagram types used to model the system during the different development phases. This paper presents an automated COSMIC-based approach for checking the consistency between the activity and component diagrams. First, it defines measurement procedures to determine the functional size of both diagrams. Secondly, it proposes a set of heuristics to ensure the consistency in terms of COSMIC-FSM. Third, it presents a tool for measuring the functional size of these diagrams, and then checking their consistency.

Keywords: Functional Size Measurement, COSMIC-ISO/IEC 19761, UML Activity Diagram, UML Component Diagram, Consistency Rules.

1 Introduction

Thanks to its various diagram types, UML provides for a multi-view representation of user functional requirements, system structure, and dynamic behavior. Nonetheless, the diversity of UML diagram types can introduce inconsistencies among the various diagrams representing the same system during the various development phases. Evidently, these inconsistencies may lead to errors, high development costs, and potentially software failures. It is therefore vital to have an approach for ensuring the consistency among the various UML diagrams modeling the same system.

To detect inconsistencies among UML diagrams, several approaches have been proposed either based on meta-modeling [8], or based on the adoption of formal methods [16]. The first category of approaches examines only the syntactic constraints among the UML concepts; the second category relies on the semantic constraints among the UML concepts and requires a certain level of expertise in the formal method used. In addition, none of them provides for a means both to *detect* potential inconsistencies and to *estimate* functional size attributes of one diagram from another already elaborated. Such a means can be offered through a measurement method. In this paper, we illustrate the feasibility of such an approach by using the functional size of software.

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 48–63, 2013.

In the software measurement literature, five measurement methods have been recognized as standards to measure the functional size of software applications: IFPUG (ISO 20926), MKII (ISO 20968), NESMA (ISO 24750), FiSMA (ISO 29881), and COSMIC (ISO 19761). The main advantage of the functional size measurement (FSM) of COSMIC is its ability to quantify software from a user's point of view independently of any quality and technical criteria. In addition, compared to other international measurement methods, COSMIC is designed to be applicable to any type of software. These advantages motivated several researchers to investigate the use of COSMIC to determine the functional size of UML diagrams.

Current proposals to use FSM for UML focused on particular diagrams, e.g., the use case diagram [15], [11], [4], [3], [5]; the sequence diagram [15], [11], [3], [5]; the activity diagram [4]; the class diagram [15], [11] and [5]; or the component diagram [12] and [11]. Except for [15] and recently [12], these proposals treated UML diagrams in an isolated way. In addition, the UML Activity Diagram (UML-AD) has not been explored in detail, despite its importance in representing behavioral aspects of software. Similarly, the UML Component Diagram (UML-CD) has not been treated in spite of its advantage in component reuse especially for the development of complex applications.

This paper has a two-fold contribution. First, it completes our previous work [15] which focused on the functional size of the UML use case diagram as a reference measurement for the FSM of the sequence and class diagrams. In this paper, we use the COSMIC method to measure the functional size of the UML-AD and UML-CD diagrams. Secondly, it proposes a set of heuristics that provides for both verifying the consistency of these diagrams in terms of functional size, and estimating a bound on the functional size of one diagram from a developed diagram. Such an estimate can be used for instance in a time/effort evaluation process. Furthermore, to avoid any error prone manual functional size measurement, we propose herein an automated functional size measurement as a tool for analyzing and consistency checking of both activity and component diagrams.

The remainder of this paper is organized as follows: Section 2 presents an overview of the COSMIC method, UML diagrams and existing proposals for COSMIC FSM of UML diagrams. Section 3 and 4 present, respectively, the proposed measurement procedure required for measuring the functional size of UML-AD and UML-CD with the proposed heuristics. Section 5 presents a tool that automates our approach and illustrates it through the "Rice Cooker" case study [6]. Finally, Section 6 summarizes the presented work and outlines some further works.

2 Related Works

2.1 Overview of COSMIC FSM

COSMIC has been accepted as an international standard ISO/IEC 19761 since 2003. It has been widely used in order to measure the software functional size, which is derived by quantifying the Functional User Requirements (FUR) [7]. FUR is a sub-set of the user requirements that explains what the software must do to satisfy user needs. COSMIC is developed as a generalization of Function Point Analysis [1]. Based on

data movements, COSMIC is designed to measure the functional size of real-time software, business application software, etc.

As illustrated in Fig. 1, COSMIC covers four types of data movements (Entry, Exit, Read, and Write). The exchange of data across the boundary between users and software components causes either an Entry data movement type (E: from a functional user to the functional process), or an Exit data movement type (X: from a functional process to the functional user). On the other hand, the exchange of data between storage hardware and software component causes either a Read data movement type (R: from a persistent storage to the functional process), or a Write data movement type (W: from a functional process to the persistent storage).

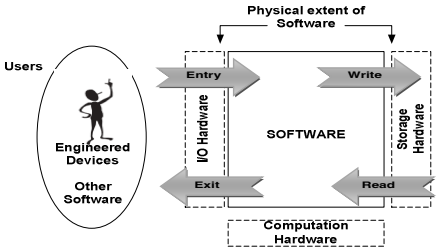


Fig. 1. COSMIC model of generic software [7]

The COSMIC measurement procedure includes three phases: measurement strategy, mapping, and the measurement. The measurement strategy phase defines the purpose and scope of measurement. In the mapping phase, the documentation of the software to be measured is analyzed and modeled to the COSMIC generic model. Finally, the measurement phase includes the application of the numerical assignment rules, where every data movement is assigned to 1 CFP (Cosmic Function Point). The software functional size is computed by adding all data movements identified for every functional process [7].

2.2 Overview of Concepts of UML Diagrams

Activity Diagrams in UML 2. Activity diagrams (UML-AD) are used to represent the dynamic and the functional aspects of a system under development. UML-AD is suitable to represent graphically the different scenarios in a use case. Its meta-model defines the different modeling elements as well as the links between them [14]. Informally, an UML-AD can be described as a directed graph of nodes and edges. There are three types of ActivityNodes: ExecutableNodes, ObjectsNodes, and ControlNodes. ActivityNodes are linked by ActivityEdges which can be control flow (linked to a control node) or object flow (linked to a node object).

To facilitate the mapping between UML-AD and COSMIC concepts, we will adopt a textual representation in the remainder of this paper. Based on the meta-model of UML 2 [14], an UML-AD **S** is defined by the following structure (1st level):

$$\boldsymbol{S} = <A, P, O, N_c, F_c, F_{obj}>$$

where:

- A: a set of system activities (ExecutableNodes)
- P: a set of its partitions (Actors)
- O: all system objects (ObjectNodes)
- N_c: all its control nodes (ControlNodes)
- F_c: all its control flows,
- F_{obj}: all its object flows.

At the 2nd level, an activity is represented by an UML-AD. Each activity includes a set of actors (represented by partition) that are responsible to realize a set of actions. An activity can include one or more pre or post condition, and one or more input or output parameters. Fig. 2 depicts a textual representation of UML-AD.

```
Number of activity: <Num of activity>
Activity: < activity name>
Partition: <Partition 1>..<Partition n>
Pre condition:([<Pre_condition 1>]..[<Pre_condition n>])
Post condition:([<Post_condition 1>]..[<Post_condition n>])
Input Parameters:([<Parameter_int 1>..<Parameter_int n>])
Output Parameters:([<Parameter_out 1>..<Parameter_out n>])
Begin: <Num> < action destination >
Body: List of actions
<Num action 1> <Name action> < Partition && [Object]>
([<Parameter_int 1>..<parameter_int n>])
([<Parameter_out 1>..<parameter_out n>])
[<Pre_condition>][<Post_condition>]
...
<Num action n> <Name of action> < Partition && [Object]>
([<Parameter_int 1>..<parameter_int n>])
([<Parameter_out 1>..<parameter_out n>])
[<Pre_condition>][<Post_condition>]
End: <Num> <Name> <source action>
```

Fig. 2. An UML-AD textual representation (2nd level)

```
Number of component: <Num of component>
Component: < component name >
--provided Interfaces--
<Interface 1> <name Interface 1>
<Operation 1> <name Operation 1>
      ([<Pre_condition 1>]..[<Pre_condition n>])
      ([<Post_condition 1>]..[<Post_condition n>])
      ([<Parameter_int 1>..<Parameter_int n>])
      ([<Parameter_out 1>..<Parameter_outn>])
…
<Operation n> <name Operation n>
…
<Interface n> <name Interface n>
--required Interfaces--
<Interface 1> <name Interface 1>
<Operation 1> <name Operation 1>
      ([<Pre_condition 1>]..[<Pre_condition n>])
      ([<Post_condition 1>]..[<Post_condition n>])
      ([<Parameter_int 1>..<Parameter_int n>])
      ([<Parameter_out 1>..<Parameter_outn>])
…
<Operation n> <name Operation n>
…
<Interface n> <name Interface n>
```

Fig. 3. An UML-CD textual representation

Component Diagrams in UML 2. Component diagrams (UML-CD) are often used for modeling complex software applications at a higher level of abstraction. An UML-CD represents the physical components of a system and all their interfaces. According to the UML-CD meta-model [14], an UML-CD is composed of a set of components that provide services through their interfaces (*provided* and *required*). Each interface can include a set of *operations*. Each operation can have one or more *pre* or *post condition*, and *input* or *output parameters*. Fig. 2 presents a textual representation of UML-CD.

2.3 COSMIC for UML

Among the researchers that studied the use of COSMIC to measure the functional size of UML diagrams, Bévo et al. investigated the mapping between concepts of COSMIC 2.0 and those of UML 1.0 [5]. Their investigation was presented through the FSM of a building access system modeled with the use case, sequence and class diagrams. Being presented through an example, it lacked the coverage of some concepts in these diagrams like the triggering event which induces one CFP. This study reports the issue of how to identify the appropriate UML concepts to represent the COSMIC functional process, and then the appropriate grain-level of measurement.

Also focussing on the UML use case, sequence and class diagrams, Azzouz et al. proposed an automated functional size measurement procedure of these diagrams when developed according to the Rational Unified Process [3]. Their tool, COSMIC-RUP, is integrated in Rational Rose. It was used to measure the functional size of two case studies, "Rice Cooker" and "Valve Control". However, the results obtained by COSMIC-RUP differ by 1 CFP from those obtained by a manual measurement for each case study. Furthermore, the proposed measurement procedure does not account for the COSMIC “system layers” concept which is important to identify the functional processes of a system under measurement.

On the other hand, Berg et al. showed that UML can be used to present FUR at four levels of refinement: Goal-level requirements, Domain-level requirements, Product-level requirements, Design-level requirements [4]. In every level, they assume that particular UML diagrams are used to model the software. In addition, they showed that the functional size can be determined using measurement methods such as Function Point Analysis (FPA) and COSMIC-Full Function Points (COSMIC-FFP) in the third level [4]. In this level, they used the use case diagram, activity diagram and class diagram. The proposed measurement approach is illustrated through a case study "The Hotel Case". Despite being the only study treating UML-AD to model the behavioral view of software, the provided measurement approach did not investiage several details of UML-AD.

Lavazza et al. studied the functional size measurement of the UML use case, sequence, and component diagrams by using COSMIC [11]. Similar to the previous works, their measurement process relies on a mapping of the COSMIC concepts onto the UML diagram concepts. It was illustrated through the FSM of the "Rice Cooker" real time software. However, the UML-AD of the "Rice Cooker" was not measured despite its usefulness in representing system details and interactions between the system and its actors.

Unlike the above works, Sellami *et al*. considered that the semantic links among the various UML diagrams of a system model must be respected in any measurement process [15]. First, they presented an approach to measure the functional size of the UML use case diagrams. Then, they propose to use the functional size of the use case diagram as a reference measurement for the sequence and class diagrams. To overcome the high level of abstraction of the use case diagrams, the authors used an intuitive documentation of the use cases proposed by Ali et al. [2]. The produced measurement can thus be used to verify the consistency of the the use case diagram with the functional size of the sequence diagrams. The proposed approach was verified using a business application "ALLOC" [9].

Also exploring the semantic links among UML diagrams, Lind et al. developed a tool "CompSize" to provide the functional size of the component diagram (UML-CD) [12]. For this, they extended UML-CD to represent necessary information. However, their measurement process defines data movements independently of the software boundary, which may lead to incorrect results.

Table 1. Summary of the proposals mapping COSMIC on UML

COSMIC concepts	Lind *et al*. [12]	Sellami *et al*. [15]	Lavazza *et al*. [11]	Berg *et al*. [4]	Azzouz *et al*. [3]	Bevo *et al*. [5]
Application boundary	Compo-nent	Use Case Sequence	Use Case Compo-nent	Use Case	Use Cases	Use Case
System layers	Compo-nent	-	-	-	-	-
Functional User	Compo-nent	Use Case Sequence	Use Case Compo-nent	Use Case	Use Cases	Use Case
Triggering event	-	Use Case	Compo-nent	Activity	Sequence	Sequence
Data group	-	-	Compo-nent Class	Class	Class	Class
Data attribute	-	-	-	-	Class	Class
Functional Process	Compo-nent	Use Case Sequence	Sequence Use Case	Activity	Use Cases	Use Case
Data Movement	Compo-nent	Use Case Sequence Class	Sequence	Activity	Sequence	-

In summary, as shown in Table 1, current researches proposed mappings between most of the COSMIC concepts and some UML diagrams. None of these studies considered *all* COSMIC concepts. In addition, some works automated their approach Lind et al. [12], Azzouz et al. [3], Bévo et al. [5]. Nonetheless, the proposed tools still require manual intervention of the user (*cf.* [3], [5]) and have a large margin of error that can reach 33% [5]. Furthermore, when examining current measuring approaches, one notice that further work is needed to explore the semantic links among the UML diagram types to provide for a confrontation/estimation/consistency verification among the different diagrams modeling a given system. In the following sections, we contribute to this research end through the UML-AD and UML-Component diagrams.

3 Measuring UML-AD

3.1 Modeling Rules

To model an UML-AD that can be measured using COSMIC, we propose 12 modeling rules. These rules are inspired from "good design practices" and are intended to eliminate certain inconstancies. In addition, our 12 modeling rules are defined to make the application of COSMIC concepts easier.

The first three rules (R1, R2 and R3) are required at the functional level whereas the remaining rules (R4 to R12) are used at the dynamic level.

- R1: Represent all system processes and the relationship between them at the functional-level.
- R2: Any component or user that interacts in the realization of a process is considered as an actor in the UML-AD.
- R3: If the activity requires incoming information or a condition that must be satisfied, it is considered as a pre-condition.
- R4: Each functional process will be represented by an activity diagram.
- R5: Each external actor (system user) is represented by a partition.
- R6: Any internal actor is represented by a partition.
- R7: All actions performed by the same actor are grouped in the same partition.
- R8: Any action requires retrieved or written data from/to a persistent storage; it must be associated to an object node that contains the data to be used.
- R9: Avoid the transitions between the actors and the system when they are intended to indicate a possible end of the functional process (failure or success).
- R10: Every guard condition is considered as a trigger event of its corresponding action.
- R11: Action data recovery and action of writing data are differentiated by the direction of the transition.
- R12: If the action requires incoming information that must be satisfied, it is considered as a pre-condition.

Note that it is required to distinguish between external actor's partition and system's partition (internal actor). This distinction can be indicated through a description of the actor's attribute.

3.2 Mapping COSMIC on UML-AD

Similar to existing approaches, to measure the functional size of an UML-AD, one needs to define the mapping between the COSMIC concepts and those of UML-AD. As listed in Table 2, the mapping deals with the identification of functional users, boundary, functional processes, etc.

Table 2. Mapping COSMIC on UML-AD

COSMIC V.3.0.1	UML-AD concepts
Functional User	<Patition> actor who interacts with the system
Boundary	Conceptual line between two partitions (system and actor)
Functional Process	<Activity> an executable activity node presented in the first level
Triggering Event	<Pre condition> of an activity <guard condition> in a decision or a fusion node <Pre condition> of an action
Persistent Storage	<Object> object node: Storage
Transient data group	<Object> object node: Pins
Entry	An incoming data between two partitions (from actor to system)
Exit	An outgoing data between two partitions (from system to actor)
Read	<Parameter_int> Read access from an object node
Write	<Parameter_out> Write access to an object node

3.3 FSM Measurement Formulas

At the functional level, an UML-AD **(A)** consists of a set of activities. Each activity is a functional process. Thus, the functional size of an UML-AD **(A)** can be measured as follows:

$$FSM(A) = \sum_{i=1}^{n} FSM(a_i) \tag{1}$$

where:

- FSM **(A)**: the functional size of the **(A)**. *n*: the number of activities in **A** (1st level).
- FSM ($\mathbf{a_i}$): functional size of the activity $\mathbf{a_i}$ (2nd level). (see Fig. 2)

At the dynamic level, an activity $\mathbf{a_i}$ consists of a set of actions $\mathbf{act_{ij}}$. According to Knieke *et al*. in this level, an activity is made by at least one action, an end node, and an initial node [10]. Thus, the functional size of an activity $\mathbf{a_i}$ is given by:

$$FSM(a_i) = FSMcond(\text{Precond } a_i) + \sum_{j=1}^{m} FSM(act_{ij}) \tag{2}$$

where:

- FSM($\mathbf{a_i}$): functional size of the activity $\mathbf{a_i}$.
- *m*: the number of actions $\mathbf{act_{ij}}$ of the activity $\mathbf{a_i}$ (2nd level).
- FSM ($\mathbf{act_{ij}}$): the functional size of $\mathbf{act_{ij}}$ of $\mathbf{a_i}$ (2nd level) calculated according to (3).
- FSMcond (Precond $\mathbf{a_i}$): FSM of the pre-condition of $\mathbf{a_i}$ calculated according to (4).

The functional size of an action $\mathbf{act_{ij}}$ is given by:

$$FSM(act_{ij}) = FSMcond(\text{Pr}econd\ act_{ij}) + FSMparam(Param\ act_{ij}) \tag{3}$$

where:

$$FSMcond\ (\text{Precond}\ act_{ij}) = \begin{cases} 1\ CFP & if\ act_{ij}\ has\ a\ \text{pre-condition} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

$$FSMparam\ \ (Param\ act_{ij}) = \begin{cases} 1\ CFP\ if\ act_{ij}\ has\ input/output\ parameters \\ 0 \quad otherwise \end{cases} \quad (5)$$

If an action is preceded by a decision or a fusion node, then the guard condition is considered as a trigger event. It is necessary to add 1 CFP to action's size.

$$FSMcond\ \ (Condgarde\ ) = \begin{cases} 1\ CFP & if\ act_{ij}\ has\ a\ guard\ condition \\ 0 & otherwise \end{cases} \quad (6)$$

When the end of an action in an Actor partition causes the execution of an action in a System partition, then the control flow corresponds to an Entry data movement. Otherwise, it is an Exit data movement. Hence,

$$FSMactTyp\ \ (actTyp\ ) = \begin{cases} 1\ CFP & if\ the\ actTyp\ is\ a\ particular\ case\ of\ actions \\ 0 & otherwise \end{cases} \quad (7)$$

4 Measuring UML-CD

4.1 Mapping COSMIC on UML-CD

Establishing a mapping between the COSMIC concepts and those of UML-CD is also needed to facilitate the measurement of the UML-CD functional size. Our mapping is inspired from the proposition of Lavazza et al. [11]. Table 3 shows our proposed mapping.

Table 3. Mapping COSMIC on UML-CD

COSMIC	UML-CD concepts
Functional User	<Component> External entity directly connected with the system components
Boundary	Frontier between two components (external and system components)
Functional Process	Set of <Operation> in one or more interfaces carrying out a process
Triggering Event	<Operation>in a system interface invoked directly by an external entity
Persistent Storage	<Component> Classes: physical components
Transient data group	<Parameter_int> <Parameter_out> Data across the system boundary, interface's operations or parameter's operations
Entry	<Operation> in a <required interface> directly connected to the system
Exit	<Operation> in a <provided interface> directly connected to the system
Read	<Operation> Get type operation in a system component
Write	<Operation> Set type operation in a system component

4.2 FSM Measurement Formulas

The functional size of the UML-CD (**C**) is given by:

$$FSM(C) = \sum_{i=1}^{n} FSM(S_i) + \sum_{j=1}^{m} FSM(I_j) \quad (8)$$

where:

- FSM (**C**): functional size of the UML-CD **(C).**
- FSM(S_i): functional size of operations in a system component S_i; it is calculated according to (9). *n:* number of the system components.
- FSM (Ij): functional size of required and provided interfaces Ij; it is calculated according to (10). *m:* number of the interfaces required and provided in **(C)**.

The functional size of operations in a system component is given by:

$$FSM(S_i) = \sum_{j=1}^{y} FSM_{op}(Op_{ij}) \quad (9)$$

where:

- FSM (S_i): functional size of operations in a system component.
- *y:* number of operations in a component system. *(i=1,...n)*
- FSMop (Op_{ij}): functional size of the operation Op_{ij}. (1CFP)

The functional size of required and provided interfaces is given by:

$$FSM(I_j) = \sum_{k=1}^{z} FSMop(Op_{jk}) \quad (10)$$

where:

- FSM (I_j): functional size of required and provided interfaces.
- *z:* number of operations in the interface I_j. *(j=1,...m)*
- FSMop (Op_{jk}): functional size of the operation Op_{jk}.

4.3 Correspondence between UML-AD and UML-CD

Equation (11) can be used to verify the consistency between an UML-AD **(A)** and an UML-CD (***C***) in terms of COSMIC FSM:

$$2 \leq FSM(C) \leq FSM(A) \quad (11)$$

Recall that an UML-AD is composed of at least one actor and a system, an initial node, an end node, and a set of actions (R4, R5, R6, and R7). In the second level of abstraction, a UML-AD represents a functional process. On the other hand, based on COSMIC concepts, a functional process (FP) is composed of two data movement (Entry and Exit or Write). Thus, the FSM of a UML-AD is at least equal to 2 CFP; that is, we should have (FSM (A) ≥ 2 CFP).

On the other hand, the FSM of an UML-CD is always less than the FSM of an UML-AD. Hence, the FSM of an UML-CD is at least equal to 2 CFP. The maximum size of an UML-CD depends on the size of the UML-AD.

Equation (11) gives a confrontation means for both diagrams in terms of COSMIC FSM. Besides this high-level FSM boundary confrontation, we propose the following five heuristics to ensure the COSMIC FSM consistency between **A** and **C**:

1. **ConsR1:** Any component in the **C** is a partition representing an actor in **A**.
2. **ConsR2:** Any method in an interface is represented by an action in a partition.
3. **ConsR3:** Input/output parameter's operations in **C** correspond to the input/output pins in **A**.
4. **ConsR4:** Object nodes in **A** are represented by class's components in **C**.
5. **ConsR5:** Pre and post-conditions of an operation in **C** correspond to pre and post-conditions of an action in **A**.

5 Example: The Rice Cooker

5.1 Using Manual Measurement

To illustrate the application of the proposed FSM formula, we use the real time software application "Rice Cooker" case study. The FURs of this case study are described in [6]. The question is how to determine the functional size of the three functional processes (FP1: Set Target Temperature, FP2: Adjust Temperature and FP3: Lamp Control) as described in [6]. These processes are triggered by three events which are respectively: Signal 30s, Signal 5s, and Tick (elapsed). Fig. 4 shows the UML-AD of "Rice Cooker" application at a high level of abstraction.

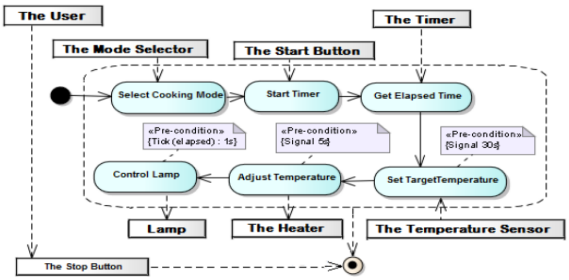


Fig. 4. UML-AD of the "Rice Cooker" application (high level of abstraction)

Fig. 5 shows a textual description of the FP1 "Set Target Temperature". This textual description can provide the FSM of FP1 by using only the mapping with COSMIC (Table 2).

```
Number of activity: <4>
Activity: <set target temperature>
Partition: <Timer> <Software Controller>
Pre condition : <Signal 30s>
Input Parameters:
Output Parameters:
Begin: <1> <Receive signal 30s>
Body: List of actions
<1> <Receive signal 30s> <Software Controller>
<2> <Get cooking mode> <Software Controller && CookingMode> <Cooking
mode>
<3> <Get elapsed time> <Software Controller>
<4> <Provide elapsed time> <Timer>
<5>      <Get      Cooking      Temp>      <Software      Control-
ler&&Temperature><Temperature>
<6>  <Set  target  temperature>  <Software  Controller  &&Temperature>
<Target-Temperature>
```

Fig. 5. Textual description of the UML-AD "Set Target Temperature"

Table 4 presents in detail the measurement results of the UML-AD for the functional process (FP1). Due to space limitation, we will present only the measurement results for the two other processes (FP2, FP3). In addition, based on the component diagram of the "Rice Cooker" in [11], which includes three components and five interfaces; we will present the FSM results of the related UML-CD in Table 5.

Table 4. Measurement results (Activity diagram of the "Rice Cooker")

FP	Application of measurement formulas for UML-AD		CFP
FP1	$FSM(FP1) = FSMcond(Signal\ 30s) + \sum_{j=1}^{6} FSM(act_{1j})$	(2)	6
	$FSMcond(Signal\ 30s)$	(4)	1
	$FSM\ act1j = FSMcond(Precond\ act_{1j}) + FSMparam(Param\ act_{1j})$	(3)	0+3
	$FSMcond\ (Precond\ act_{1j}) = \begin{cases} 1\ CFP & \text{if } act_{1j} \text{ has a precondition} \\ 0 & \text{otherwise} \end{cases}$	(4)	0
	$FSMparam\ (Param\ act_{1j}) = \begin{cases} 1\ CFP & \text{if } act_{1j} \text{ has input/output parameters} \\ 0 & \text{otherwise} \end{cases}$	(5)	3
	$FSMactTyp\ (actTyp) = \begin{cases} 1\ CFP & \text{if actTyp is a particular case} \\ 0 & \text{otherwise} \end{cases}$	(7)	2
	$\sum_{j=1}^{6} FSM(act_{1j}) = FSM(Receive\ signal\ 30s) + FSM(Get\ cooking\ mode) + FSM(Get\ elapsed\ time) + FSM\ (Provide\ elapsed\ time) + FSM(Get\ cooking\ temp) + FSM\ (Set\ target\ temperature)$		0+1+ 0+0+ 1+1
FP2	$FSM\ (FP2) = FSMcond(Signal\ 5s) + \sum_{j=1}^{7} FSM(act_{2j})$	(2)	6
FP3	$FSM\ (FP3) = FSMcond(Signal\ 1s) + \sum_{j=1}^{3} FSM(act_{3j})$	(2)	2
Total	$FSM(A) = \sum_{i=1}^{3} FSM(a_i)$	(1)	14

According to equation (11), it can be ensured that the UML-AD design is conformed to the UML-CD design. In addition, assuming that the consistency heuristics are satisfied, the FSM difference between the UML-CD (12 CFP) and the UML-AD

(14 CFP) can be justified by the difference in the levels of abstraction. Since UML-AD represents software at a more detailed level and UML-CD represents software at a high-level of abstraction, UML-CD does not represent all software details as well as UML-AD. Looking closely, in the UML-AD FP2, the extra CFP is due to the guard conditions which are not represented in the UML-CD.

Compared to existing works, our measurement results are consistent with those of Lavazza et al. [11] and they ensure the correctness of our measurement procedures. Albeit, there are some distinctions in the FSM results; for instance, we measured 14 CFP for UML-AD and 12 CFP for UML-CD of the "Rice Cooker" case study, while the FSM of the same case study calculated by [11] is equal to 11 CFP. However, their value is provided according to the identification of data movement involved in the functional processes. In other words, it is independent of the UML diagrams.

In addition, it can be observed that, for the UML-AD, there are extra 3 CFP for three "Exits". Because FP1 contains the transition "Get elapsed time", it should be considered as a data movement type "Exit". However, [11] ignored this data movement. In FP2, the extra 2 CFP are due to reasons: *(i)* the guard conditions which were not treated by [11] for both actions "Start heater" and "Stop heater". They considered the command "HeaterOn and HeaterOff" as 1 data movement "Exit"; and *(ii)* the action "Get Actual Temperature" was not identified by [11] since they considered the "Actual Temperature" to be returned by "Temperature Sensor" following the demand of "Software Controller".

Table 5. Measurements results for the UML-CD of the "Rice Cooker"

Application of measurement formulas for UML-CD		CFP
$FSM(C) = \sum_{i=1}^{3} FSM(S_i) + \sum_{j=1}^{5} FSM(I_j)$	(8)	12
$FSM(S_1: CookingModeC) = \sum_{j=1}^{2} FSM_{op}(Op_{1j}) =$ FSM(GetMode():Cooking_mode) +FSM(SetMode(mode:Cooking_mode))	(9)	1+1
$FSM(S_2: CookingSpecsC) = \sum_{j=1}^{1} FS_{op}(Op_{2j}) =$ FSM(GetCookingTemp (time: Integer, mode: Cooking_mode): Integer)	(9)	1
$FSM(S_3: CookingStateC) = \sum_{j=1}^{2} FSM_{op}(Op_{3j}) =$ FSM(SetTargetTemp(temp)) + FSM(GetTargetTemp():Integer)	(9)	1+1
$FSM(I_1: TimedEvents) = \sum_{j=1}^{3} FSM_{op}(Op_{1i}) =$ FSM(Signal 30s()) + FSM(Signal 5s())+ FSM(Tick(elapsed))	(10)	1+1+ 1
$FSM(I_2: TempSensorCommands) = \sum_{j=1}^{1} FSM_{op}(Op_{2i})$ =FSM(ReadTemp (): Integer)	(10)	1
$FSM(I_3: HeaterOnInterface) = \sum_{j=1}^{1} FSM_{op}(Op_{3i}) =$ FSM(HeaterOn())	(10)	1
$FSM(I_5: LampCommands) = \sum_{j=1}^{1} FSM_{op}(Op_{5i}) =$ FSM(On())	(10)	1

Furthermore, for the UML-CD, the extra 1 CFP is due to the operation "Set-Mode(mod:Cooking_mode)". Indeed, this operation corresponds to another functional process (stop cooking). If we take into account the "scope" according to COSMIC method, this operation will not be considered. In addition, our measuring scope is limited by the three FP (Set Target Temperature, Adjust Temperature and Lamp Control).

5.2 Using Consistency Checking Tool

The automation of the measurement procedures overcomes various measurement problems induced by manual, error prone procedures. We proposed a tool that automates consistency checking between UML-AD and UML-CD based on equation (11). The current version of our tool is implemented in java JDK 1.6 using Eclipse SDK 3.7.1.

This tool can help us to calculate the functional size of both UML-AD and UML-CD and to check the consistency between them in terms of their functional size. If the obtained result does not respect the margin size presented in the equation (11), then each of the UML-AD or UML-CD is more fault-prone, and can lead to failure. In this case, the tool alerts the software developers/designers with the presence of any modeling error in the design phase. Thus, the margin size expressed in CFP units can be used to predict if the transformation from one diagram to another is properly done.

Fig. 6 presents the main GUI of our consistency checking tool. It accepts UML diagrams produced through the papyrus plugin[1]. After measuring a diagram, the tool can produce the results in a tabular format (see Fig. 6) for the "Rice Cooker" example. The tool also provides the results of the consistency checking analysis of the UML-AD and UML-CD.

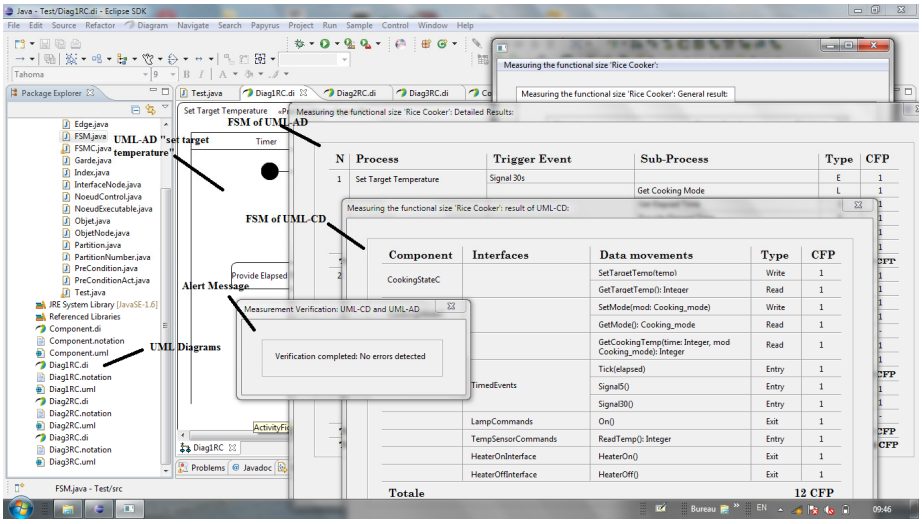


Fig. 6. Consistency checking tool

6 Conclusions

Applying COSMIC FSM method in the design phase for automated consistency checking between activity diagram (UML-AD) and components diagram (UML-CD) is the main purpose of this paper. Here, consistency was examined in terms of the functional size of these diagrams. Measuring the functional size of UML-AD and

[1] http://www.eclipse.org/papyrus/

UML-CD is useful not only in identifying modeling errors but also in automating design quality analysis.

Our functional size measurement procedures for UML-AD and UML-CD were defined based on the mapping between COSMIC concepts and those of UML diagram concepts. In addition, we proposed a measurement interval that can be used as a guideline by designers to verify the consistency between these two diagrams and to identify modeling errors. We have also proposed a set of modeling rules to ensure the consistency between these diagrams. Finally, we have illustrated the proposed measurement procedures by using the "Rice Cooker" case study, and confronted our measurement results with those of [11]. This case study was conducted through a tool that implements our measuring and consistency checking approach.

Further works including the use of measurement results of UML-AD and UML-CD should be investigated. Indeed, it would be interesting to examine how these measures can help software managers and leaders to complete their project within the scheduled dates. The proposed formulas need to be applied on larger case studies to ensure the quality of our measurement procedures. Furthermore, the proposed automated tool can be extended to include other UML diagrams such as use case, sequence, etc. and used to get more realistic estimation for managing software project.

References

1. Albrecht, A.J.: Measuring application development productivity. In: Proceedings of the IBM Application Development Symposium, Monterey, California, pp. 83–92 (October 1979)
2. Ali, M., Ben Abdallah, H., Gargouri, F.: Validation des besoins dans les modèles UML 2.0. In: XIVème congrés INFORSID, Hammamet, Tunisia (2006)
3. Azzouz, S., Abran, A.: A proposed measurement role in the RUP and its implementation with ISO 19761: COSMIC-FFP. In: SMEF 2004, Rome, Italy (2004)
4. van den Berg, K., Dekkers, T., Oudshoorn, R.: Functional size measurement applied to UML-based user requirements. In: SMEF 2005, Rome, Italy (2005)
5. Bévo, V., Levesque, G., Abran, A.: Application de la méthode FFP à partir d'une spécification selon la notation UML. In: IWSM 1999, Lac Supérieur, Canada (1999)
6. COSMIC Group. Case Study: Rice Cooker (May 22, 2008)
7. COSMIC Functional Size Measurement Method Version 3.0.1, Measurement Manual, Published by the COSMIC Group (2011), http://www.cosmicon.com/portal/dl.asp
8. Engels, G., Heckel, R., Küster, J.M.: Rule-Based Specification of Behavioral Consistency Based on the UML Meta-model. In: 4th International Conference on UML, Modeling Languages, Concepts, and Tools, London, UK, pp. 272–286 (2001)
9. Gabay, J., Gabay, D.: UML 2 Analyse et conception: mise en œuvre guidée avec des études de cas. Dunod, Paris (2008)
10. Knieke, C., Huhn, M., Lochau, M.: Modeling and Validation of Executable Requirements Using Live Activity Diagrams. In: SERA 2008, Prague (2008)
11. Lavazza, L., Del Bianco, V.: A Case Study in COSMIC Functional Size Measurement: The Rice Cooker Revisited. In: Abran, A., Braungarten, R., Dumke, R.R., Cuadrado-Gallego, J.J., Brunekreef, J. (eds.) IWSM 2009. LNCS, vol. 5891, pp. 101–121. Springer, Heidelberg (2009)

12. Lind, K., Heldal, R., Harutyunyan, T., Heimdahl, T.: CompSize: Automated Size Estimation of Embedded Software Components. In: IWSM 2011, Nara, Japan (2011)
13. Luckson, V., Lévesque, G.: Une méthode efficace pour l'extraction des instances de concepts dans une spécification UML aux fins de mesure de la taille fonctionnelle de logiciels. In: ICSSEA 2004, Paris (2004)
14. OMG Unified Modeling Language (OMG UML). Version 2.4.1. Object Management Group (2011)
15. Sellami, A., Ben-Abdallah, H.: Functional Size of Use Case Diagrams: A Fine-Grain Measurement. In: ICSEA 2009, Porto, Portugal (2009)
16. Sengupta, S., Bhattacharya, S.: Formalisation of UML Diagrams and Their Consistency Verification – A Z Notation Based Approach. In: Isec 2008, Hyderabad, India (2008)

An MDE Approach to Develop Mobile-Agents Applications

Tahar Gherbi[1], Isabelle Borne[1], and Djamel Meslati[2]

[1] IRISA Laboratory, South Brittany University, Vannes, France
{tahar.gherbi,isabelle.borne}@univ-ubs.fr
[2] Department of Computing, University of Annaba, Annaba, Algeria
meslati_djamel@yahoo.com

Abstract. The complexity and scope of software systems continue to grow. One approach to deal with this growing complexity is to use intelligent multi-agents system. Agents may be stationary or mobiles. Our work contributes to bridge the gap between agent oriented software engineering methodologies and mobile-agent systems. Indeed, we aim to propose an approach to develop multi-agents systems including mobile agents. This paper focuses principally on our design meta-model. Therefore, it gives an overview of our approach, discusses the issue of mobile-agents platforms compliance with MASIF and FIPA specifications; then examines our design meta-model versus particularly three works supporting mobility by extending a multi-agents system methodology (MaSE, Gaia, and AALAADIN).

Keywords: Mobile Agent, MAS, MaSE, m-Gaia, AALAADIN, PIM, MDE.

1 Introduction

Mobile agents are a promising paradigm to design and implement distributed applications. They have known considerable enthusiasm in the research community, although they have not been translated into a significant number of real-world applications.

Research on mobile agents has been underway for over a decade, particularly in the areas of network management and electronic commerce. Then with, among others, the rapid development of wireless networks, the spread of mobile devices using networks, the development of new networks (such as Wireless Sensor Networks) and the innovation in the field of Cloud Computing, there was an increase in the use of mobile agents. Applications based on mobile agents are being developed in industry, government and academia; and experts predict that mobile agents will be used in many Internet applications in the coming years [31].

A mobile agent is a software agent that can, during its execution, move from one site to another, to access data and/or resources. It moves with its own code and data, but possibly with its execution state also. The agent decides independently about its movements. Therefore, mobility is controlled by the application itself and not by the runtime system as is the case of processes migration in operating systems.

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 64–80, 2013.

There are no specific applications for mobile-agents [29]. In fact, mobile agents are likely to complete or replace the traditional paradigms of client-server architecture, such as message passing, remote procedure call, remote object invocation and remote evaluation. Thus, any application made with mobile agents can be made with any traditional paradigm. The use of mobile agents is, however, advantageous in heterogeneous and dynamic environments that are the trend of modern Internet applications [10]. Indeed, mobility is of great interest for applications whose performance varies depending on the availability and quality of services and resources, as well as the volume of data moved over network links subject to long delays or disconnections; and applications running on ad hoc networks, or including mobile devices. However, mobility is not an interaction as an agent does not need to be mobile to communicate. This motivated the inclusion of the mobility model in the design phase [34]. Indeed, the development of mobile-agents applications was generally done without considering the mobility aspect in the analysis and design phases. It was often treated in the implementation phase [6]. Including this aspect in the analysis and design phases allow for a better design of this kind of applications: this gives to the designer the ability to use mobility to fulfill the goals of his mobile-agents application [32].

For this reason, we have presented in [20], our meta-model to design MAS (Multi-Agents Systems) including mobile agents and we have discussed it versus some formalisms extending UML (Unified Modeling Language) for mobile agents modeling. In this paper we summarize, in section 2, the different approaches for mobile-agents modeling. Section 3, presents an overview of our approach to develop MAS including mobile-agents and discusses the issue of mobile-agents platforms compliance with MASIF (Mobile Agent System Interoperability Facility) and FIPA (Foundation for Intelligent Physical Agents) specifications. Section 4, explains the choices that have guided the construction of our design meta-model. Section 5 presents a case study and situates our design meta-model versus particularly three works extending MAS methodologies to support mobility: [32] extending MaSE (Multiagent Systems Engineering), [34] extending Gaia and [27] extending the AGR (Agent, Group and Role) meta-model of AALAADIN, which is a part of our design meta-model. Section 6 concludes the paper and presents some perspectives.

2 Related Works

According to [25], mobile-agents applications modeling can be done by three approaches: design patterns approaches, as in [3] and [26], formal approaches, as in [30], and semi-formal approaches, in which we distinguish two classes [4]: formalisms extending UML notations, as in[1] [6], [13], [24] and [25], and approaches extending a MAS methodology, as in [32], [34] and [27].

Weary of inventing and re-inventing solutions to recurrent problems, agent design patterns can help by capturing solutions to common problems in agent design [3]. However, design patterns have fields of action which are more or less restricted and need to be known. In addition, most of mobile-agent design patterns presented in

[1] Other formalisms were discussed in [20].

literature are difficult to apply in practice due to the lack of a suitable approach to identify, document and apply them [26]. Formal approaches are good in formalizing simple systems, but for large systems a visual notation is needed to easily grasp the specifications and to specify the system from different points of view [5]. Therefore, we are interested in semi-formal approaches.

Most of works on semi-formal approaches propose formalisms extending UML. Some address only one aspect of mobility, such as the mobility path, as in [24]; some fix the set of sites where the agent can move, as in [6]; some include details from MASIF, as in [6], or from FIPA standard for interaction, as in [13]. These formalisms are useful, good contributions and sources of inspiration. Belloni et al. suggest in [6] to work more on methodological aspects, by exploring how an existing software development process can be extended to incorporate notations. They recommend the exploration of the Unified Process which seems to be the most appropriate. However, to contribute in bridging the gap between AOSE (Agent Oriented Software Engineering) methodologies and mobile-agent systems, as suggested in [29] and realized in [32], [34] and [27], we are interested to extend a MAS methodology. Indeed, a separation between the community of MAS and intelligent agents on one side and the community of mobile agents on the other side was raised and discussed in [29]. This separation resulted from the fact that researchers in the MAS community come mainly from the artificial intelligence field, while those in the mobile-agents community come primarily from the distributed systems (even operating systems) field. They use different languages, have different goals and different ways to view problems: the MAS community tries to solve conceptual problems (subject to analysis and simulations based on distributed computing), where the mobile-agents community attempts to implement adequate, efficient and secure platforms. The opinions of researchers interviewed in [29] have mostly converged to a synergy between these two fields. Merging these two fields provides more capacity to solve complex problems in distributed computing becoming increasingly mobile [32].

Only few works on semi-formal approaches extend a MAS methodology to support mobility. We have encountered three in literature: [32], [34] and [27].

Self et al. [32] have extended the MaSE methodology. Fig. 1 presents a graphical overview of MaSE which consists of two phases and several steps. The progression over steps occurs with outputs from one step becoming inputs for the next. The result of the MaSE analysis phase is a set of roles that agents may play, a set of tasks that define the behaviors of specific roles, and a set of coordination protocols between those roles. The design phase models consist of agent classes, communications defined between them and components they contain. Typically, tasks from the analysis phase are transformed into components in the design phase. These, possibly multiple, components define the internal agent architecture. To support mobility, they have added in the analysis phase a *move* command (to be used it in *Concurrent Task Diagrams* describing the behaviors of *Concurrent Tasks*), and in the design phase, *mobile components* that allow the specification of activities resulting from the *move* command. Consequently, an agent is composed of components which are stationary or mobile (a mobile component contains at least one *move* activity). To control and coordinate these components, each agent contains an *Agent Component*, which fulfils also much of the agent mobility functions.

Sutandiyo et al. [34] have criticized the extended MaSE as it does not distinguish conceptually between mobile and stationary agents (even if it does it at the components' level), and because it extends the object-oriented approach rather than starting with a "pure" multi-agents background. They have proposed (Fig. 2) m-Gaia (mobile Gaia), which distinguishes between mobile and stationary agents in the *Agent model* and defines three role types (system, interface and user) in the *Role Model*. In addition, a *mobility model* was added; it manages concepts of *place types* (locations), *atomic movement* (the smallest granularity movement required to accomplish the assigned task) and *travel path* (a combination of atomic movements). The agent's moves occur at the end of *atomic movements*.

Mansour et al. [27] note that the existing meta-models and methodologies do not provide any organizational solution for designing and administrating mobile agents in an agent society, and propose MAGR (Mobile AGR) to support the agent's mobility at the organizational level. MAGR enriches the AGR (Agent, Group, Role) meta-model with the concepts of place, mobile agent and persistent role (Fig. 3). A place represents a group joined by only mobile agents; it proposes to them necessary services to move and perform actions. Agents join groups to play roles. When a mobile agent plays a role, it specifies if it is persistent or not. When it moves, all skills associated to a persistent role stay available; however, it will be automatically deleted from any list of agents playing a non-persistent role in the place.

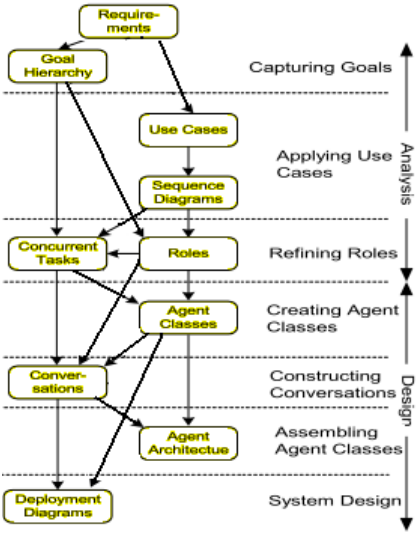


Fig. 1. MaSE methodology [32]

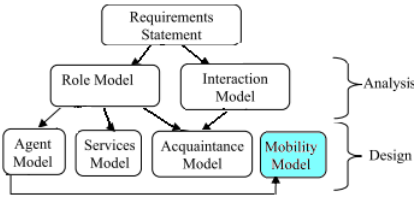


Fig. 2. Structure of m-Gaia's models [34]

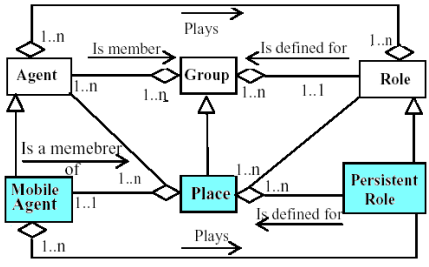


Fig. 3. MAGR meta-model [27]

In presence of mobility, MAGR's meta-model deals with the social aspect of agent's life cycle. With m-Gaia and the extended MaSE, when an agent moves, nothing is done at organizational level. Indeed, the role concept is not used after the analysis phase in both methodologies. In addition, social aspects (group, organization) are not clearly defined in MaSE, unlike organizational rules or conversations; and the

developed architectures are static[2]. Similarly in Gaia, the organization and services offered by agents are clearly static in time, as there is no hierarchical presentation. [7]

Finally and according to [2] and [22], MDE (Model Driven Engineering) helps in bringing the gap between MAS's methodologies (as the majority does not include the implementation phase[3]) and platforms[4]. However, we have not encountered any approach based on MDE and extending a MAS methodology to support mobility. Indeed, MaSE uses RUP (Rational Unified Process), m-Gaia uses the cascade model and MAGR does not propose an elaborated process[5]. Thus, our goal is to propose an MDE methodology to develop mobile-agents applications. The choice of MDE is justified also by its benefits: know-how durability, productivity gain and heterogeneous platforms consideration [9]. This explains its adoption in many works on various fields, including MAS, as in MDAD [23], ASPECS [12] and ASEME [33]. In addition, using MDE may facilitate the mobile agent moves across heterogeneous platforms: rather than sending the agent's code, we send its model which can be transformed into code on target sites.

3 Overview of Our Approach

Our design meta-model (Fig. 5) serves as a PIM (Platform Independent Model) meta-model in our MDE methodology to develop MAS including mobile-agents, which steps are shown in Fig. 4. The process starts by modeling the application as a PIM, conform to our PIMM (PIM Meta-model). Then this PIM is transformed (using transformation rules) into a PSM (Platform Specific Model), conform to the PSMM (PSM Meta-model) of the used agents' development platform. Finally, the application's code is generated from the PSM (using code generation rules).

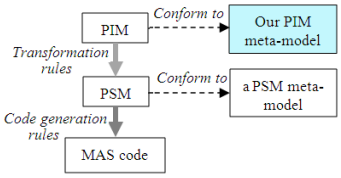


Fig. 4. MDE development process[6] for MAS

[2] O-MaSE (Organization-based MaSE) [14], an extended version of MaSE, defines a meta-model for agents to adapt their organization during execution.

[3] Meta-models in Gaia and AGR are generic: i.e., they make abstraction on the internal architecture and behavior of agents. The passage to the implementation phase remains informal and manual. [22].

[4] MAS methodologies and MAS platforms represent generally multi-agents concepts differently. [22].

[5] AGR can be seen as complementary to other agents centered methodologies, because it is insufficient alone to represent all aspects of multi-agents [22]. Indeed, MAGR (as AGR) does not provide meta-models for agents, roles and environment (i.e., domain).

[6] A CIM (Computation Independent Model) meta-model is not proposed and is left as perspective.

When elaborating a PSMM for an agents' development platform, one can ask if a common PSMM exists for agents' platforms. In fact, with the increased number of agents' development platforms, the need for standards was quickly felt. Two main agents' development standards have emerged: FIPA and MASIF. These standards provide a set of specifications and guidelines for developers (manufacturers) of agents' platforms. However none of them covers the full set of features of software agents and can rather be seen as complementary: MASIF support mobility without providing a communication language between agents and FIPA provides a communication language between agents without supporting mobility [21].Amor et al. [2] propose to use the agent model of Malacca [1] (which is based on a modular approach) as a PSMM for an MDE approach to develop MAS. One of the features of this PSMM is the ability to run a Malaca agent on any FIPA compatible agents' platform. Gervais et al. [18] propose an UML profile for mobile-agent platforms compatible with MASIF; which can be also used as PSMM. We think it will be interesting (as perspective) to propose a PSMM for agents platforms by examining the works[7] on the integration strategies combining both FIPA and MASIF standards.

This paper discusses the choices that have guided our PIMM construction. The transformation and code generation phases are the subject of future papers.

4 Choices That Have Guided Our Meta-model Construction

Choosing a MAS methodology is difficult [2] [22]. In the absence of a consensus on a meta-model to design MAS (despite the unification efforts of well-known MAS meta-models, as in [11] and [8]), we have looked for a meta-model which is simple to use, modular and evolutive, in order to extend it and support the mobility of agents. Our choice fell on the PIMM of MDAD (Model Driven Agent Development) [23] for several reasons. Firstly, it is based on the AEIO decomposition (from the VOYELLES approach [16]) which considers a MAS as composed of four bricks (or vowels A,E,I,O): Agent, Environment (i.e., domain), Interaction and Organization. This provides modularity at the models' level, rather than at the level of agents and agent's skills. The ability to interchange and reuse models of each brick has a strong potential for reuse and versatility, as there is no presupposition to use a particular model a priori [23]. Secondly, its organizational meta-model, based on AGR, does not imposes constraints on the agent internal architecture, its behavior, or its capabilities. Thirdly, MDAD is already a model driven methodology illustration for the stationary-agents applications development.

Inspired from the related works, we have enriched its PIMM with the stereotypes (Fig. 5): «MobileAgent», «Site», «Migration» (to prepare the agent before calling the Jump Action), «Jump» (to move effectively the agent to another site), «Clone» and «AfterMigration» (to integrate correctly the agent in the MAS, after its move to a new site). The concepts in gray boxes, the two associations between «SendMessage» and «ReceiveMessage» (added to ease code generation [20]), the *transferable* tagged-value in the «DomainConcept» stereotype, and the *stop* tagged-value in the «Role» stereotype are those we have added. According to Fig. 5, a group contains several roles. To play a role, the agent (stationary or mobile) must join the group containing

[7] A list of works is referenced in [21].

this role, and ask for authorization. We assume that the agent determines when it is necessary to move. However, other agents, or the agents' platform, may advise the agent to move (for example, for shutdown, load balancing, etc.); in this case, the agent's autonomous nature allows it to determine if it will actually move (section 4 gives guidelines to help treating this case). We also assume that the agents' platform handles the effective move of agents: when it receives an agent's move request (generated from the «Jump» action), it terminates the agent and sends it to the destination platform where it is restored.

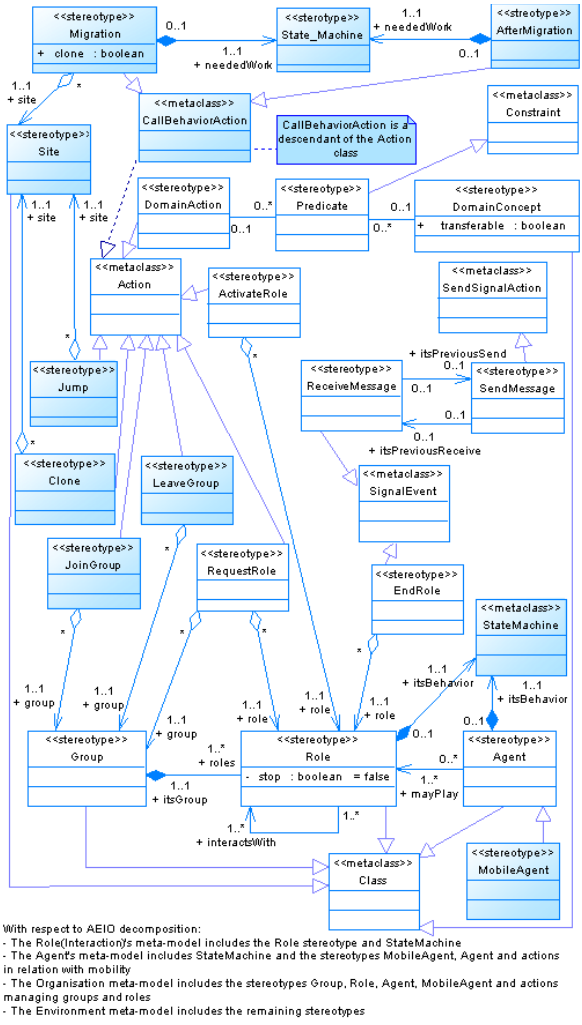


Fig. 5. PIMM for MAS including mobile agents

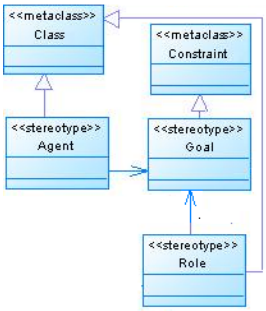


Fig. 6. Goals modeling in MDAD

Unlike MDAD, agents' and roles' goals are not expressed explicitly, but implicitly via theirs behaviors (they can even be noted as comments). However, if an explicit expression is needed, one can use for example OCL (Object Constraint Language)

constraints as in MDAD (Fig. 6). Also, we describe behaviors with state-charts diagrams, as in [32] and [25], to save transformation effort (as we use state-charts diagrams to model behaviors at PSM level also)[8].

Compared to the published version in [20], we have replaced the *AcceptEventAction* meta-class by the *SignalEvent* meta-class. In addition, we have added the «MobileAgent» stereotype to distinguish between stationary and mobile agents and have a direct mapping from PIMs to PSMs (indeed, if some mobile-agents platforms, like JavAct, do not make this distinction, others like Grasshopper, do). We have also added an association between «Clone» and «Site» stereotypes to allow flexible cloning independently of migration. The clone concept, which importance was mentioned in [32], was not considered in the extended MaSE, m-Gaia and MAGR. Finally, we have added a *stop* tagged-value (with *false* as default value) in the «Role» stereotype to be used by the agent, before it moves, to end roles held in parallel (see section 5).

In some related works, the mobile-agent itinerary is modeled to capture its movements' path, as in [6], or to describe its mission by defining tasks to do on each site in the itinerary, as in [34], [13] and [25]. We do not model this, because mobile-agents platforms normally maintain information on agents-movements' path (which can be requested) and the agent's mission is described via its behavior. We also do not fix the set of sites where a mobile agent can move, as in [6]: we assume that agents are intelligent enough to sense their environment and discover sites where they may (if necessary) move. Otherwise, the model may become unreadable in presence of lot of sites; in addition, sites are not usually all known for all applications at the design phase (e.g. in ad-hoc networks). Finally, we encourage local communications between agents; hence, we support only non-persistent roles. Consequently before leaving a site, a mobile agent must release all held roles, as in [13]. The persistent roles of MAGR generate distant communications: indeed, queries for a service provided by a persistent role will be relayed to a mirror agent representing the mobile agent playing this role. Knowing that one of the mobility's goals is to reduce the network traffic, is it really efficient for a requesting agent to see its requests relayed to a mirror agent residing on a remote site (the mobile-agent native site) rather than interacting with the concerned mobile agent by sending messages directly to it or by moving up to it?

5 Case Study

Consider (Fig. 7) a simple library database distributed on site1, site2 and site3. On each site, a stationary agent (*Librarian*) deliver the list of all books stored locally. Using a laptop, we create on site1 a mobile agent (*MobileBookSeeker*) to search for the locations of a given book over a given itinerary (e.g. site1, site2 and site3); then the laptop can disconnect. The mobile agent visits all sites, asks on each one for the local books list and filters it to check if it contains the searched book. When it finishes, it moves to its final destination (the laptop when connected) to deliver its results. A PIM for this example is given in figures 8 to 13. According to Fig.8, the *LibraryManagement* group contains three roles. The *Librarian* agent can play the *BooksListDeliver* role; and the *MobileBookSeeker* agent can play, on each visited site, the *BookChecker* role which interacts with the *BooksListDeliver* role to get the local

[8] To model behaviors, MDAD uses, at PIM level, activity diagrams and, at PSM level, ATN (Augmented Transition Network); thus, it defines transformation rules for behaviors.

books list. When *MobileBookSeeker* finishes its mission, it plays the *ResultsDeliver* role to deliver the repositories list of the searched book. Each agent (or role) has an attribute *itsBehavior* (not represented in Fig. 8 for a better readability) pointing to a state-chart diagram describing the agent (or role) behavior (in a separate figure).

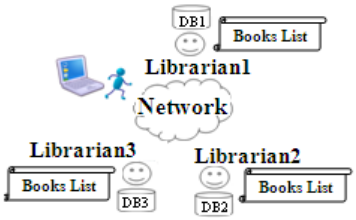


Fig. 7. A book searcher application example

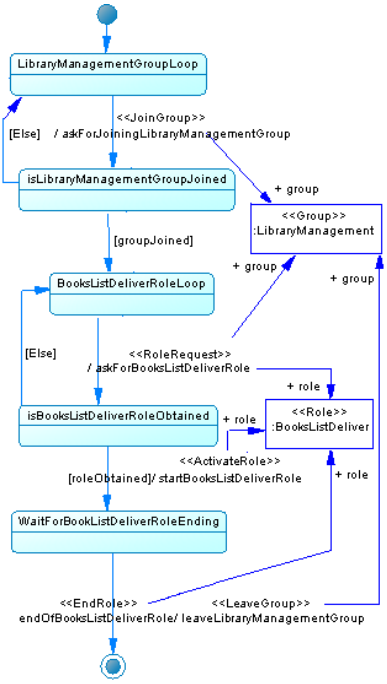


Fig. 8. Librarian behaviour

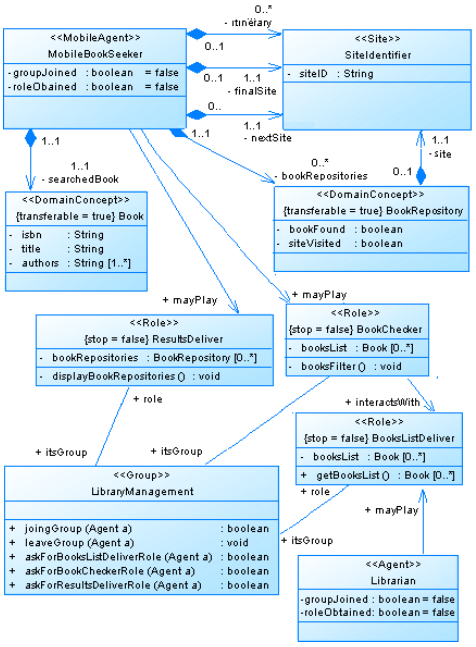


Fig. 9. The classes diagram for the example

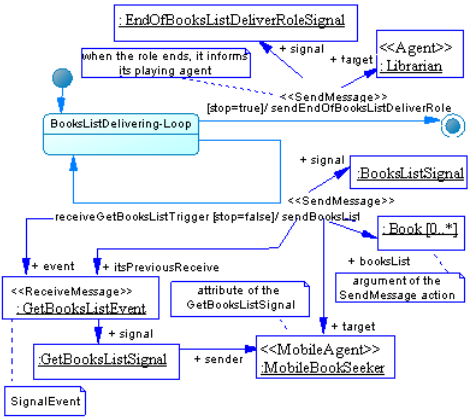


Fig. 10. BooksListDeliver behaviour[9]

[9] For details on *SendSignalAction*, *SignalEvent* and *Trigger*, see the OMG's specification (v2.3) of UML Superstructure at pages 229, 443, 442 (respectively).

The behaviors of the *Librarian* agent and the *BooksListDeliver* role are given in Fig. 9 and Fig. 10 respectively. The *Librarian* agent joins (Fig. 9) the *LibraryManagement* group, asks to play the *BooksListDeliver* role and leaves the group when the role ends. When playing the *BooksListDeliver* role (Fig. 10), it waits unlimitedly (while *stop=false*) for requests of its local books list. When (*stop=true*), the role informs its playing agent about its end. The behaviors of the *MobileBookSeeker* agent, the *BookChecker* role, and the *ResultsDeliver* role are given in Fig. 11, Fig. 12 and Fig. 13 respectively. The *MobileBookSeeker* agent joins (Fig. 11) the *LibraryManagement* group, then checks if its mission is terminated. If yes, it plays the *ResultsDeliver* role and leaves the group when the role ends; else, it plays the *BookChecker* role, then moves to the next site in the itinerary.

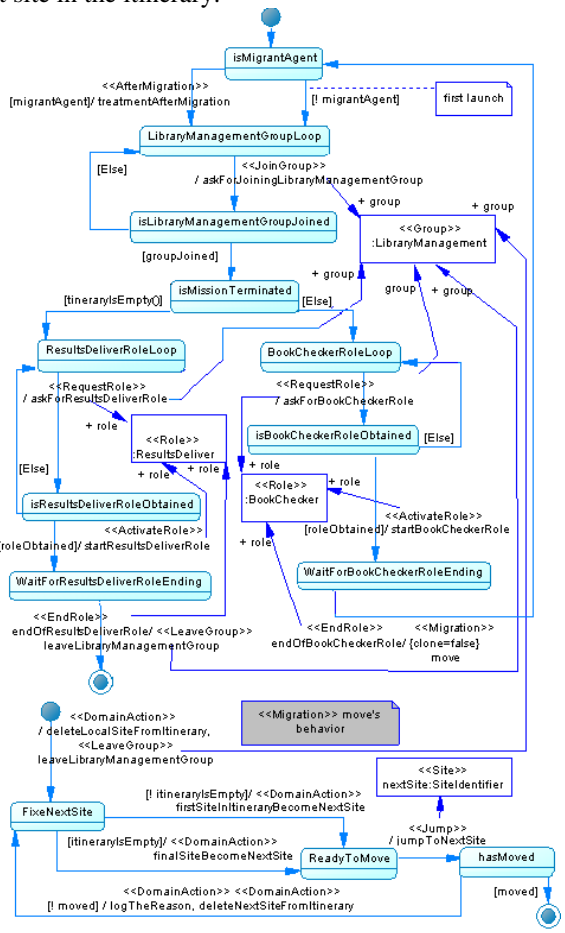


Fig. 11. MobileBookSeeker behavior

Migration and *AfterMigration* actions have their own behaviors (state-chart diagrams), where the designer can include the actions that he judges necessary. For our example, the *Migration* action leaves the group, determines the next site, and jumps to it; where the *AfterMigration* action does nothing. *Migration* and *AfterMigration*

actions may become complex, for example in the case where a mobile agent plays several roles in parallel. The agent may inside the *Migration* action ask the currently held roles to stop, wait for them to end, note from the stopped services (furnished by these roles) those it judges necessary for its activity after the move, and leaves the groups of held roles. Inside the *AfterMigration* action, the agent may search, as described in [28], for roles furnishing the noted services, joins their groups and plays them. To stop a role, its *stop* tagged-value must be made to *true*; and inside its behavior, this tagged-value must be checked to know if the role can continue or if it must end.

Moves requested by an external entity (another agent or the agents platform), can be supported, for example, by adding an *externalMoveRequest* tagged-value in the «Agent» stereotype (with *false* as default value). An external entity can request an agent to move by setting this tagged-value to *true*. When entering in any state (in its state-chart diagram representing its behavior), the agent checks this tagged-value: if it is *true*, its saves the name of the current state[10] and launches the *Migration* action. The *AfterMigration* action terminates by restoring the agent into the saved state.

When playing the *BookChecker* role (Fig. 12), the agent sends a *sendGetBooksList* message, waits to receive the books list, then checks if it contains the searched book. When playing the *ResultsDeliver* role (Fig. 13), the agent delivers its results. When a role ends it informs its playing agent.

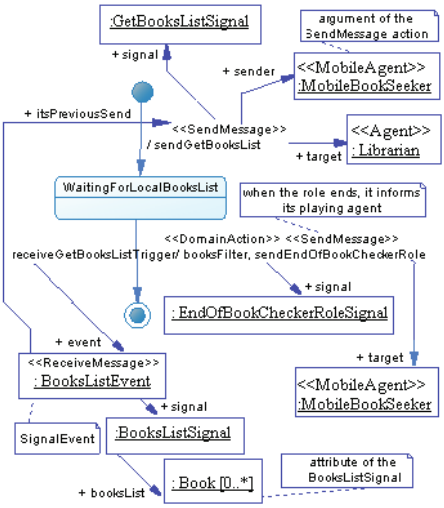


Fig. 12. BookChecker behaviour

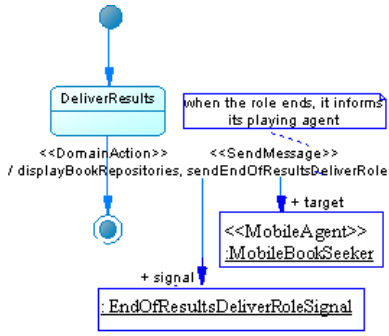


Fig. 13. ResultsDeliver behaviour

Section 3 has discussed the similarities and differences between our PIMM and other works. To see this in practice, let us model the same example using the studied methodologies. We recall that we interest only to the mobility modeling.

Using the extended MaSE, the modeling of our example, produces the agent classes in Fig. 14 (showing the roles played by agents), and the roles diagram in

[10] Or the name of the next state if the current state serves to wait for the end of a role (i.e., if its name has the form *WaitForrolenameRoleEnding*).

Fig. 15 (showing the association between roles and the concurrent tasks *searchBook*, *deliverResults*, and *deliverBooksList*). We present only the concurrent task diagram for the *searchBook* task (Fig. 16), and its corresponding mobile-component (Fig. 17). The task begins (Fig. 16) by testing if the mission is completed. If yes, it sends a *missionCompleted* message to the *ResultsDeliver* role. Else, it sends a *getBooksList*() message to the *BooksListDeliver* role, waits for the local books list, checks if it contains the searched book (actualizes eventually the repositories list), then tries to move to next site. In the *identifyNextSiteAndMove* state (Fig. 17): when a mobile component wants to move, it saves its state, informs its *Agent component* and waits for its decision. If the Agent component refuses, it replies by a *moveDenied* response; else it terminates the mobile component and orders all other components to save their states and send them to it. Every time it receives a state, it terminates the sender component. The *Agent Component* terminates, when all components terminate. Then the agent moves with all components and theirs saved states. At the target site, the *Agent Component* restarts all components and communicates their saved states to them. The *restoreState* state identifies the state in which the component restarts after migration; in the case of the *searchBook* task, the component restarts always in the *isMissionCompleted* state.

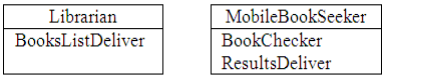


Fig. 14. Agent classes

Fig. 15. Roles diagram

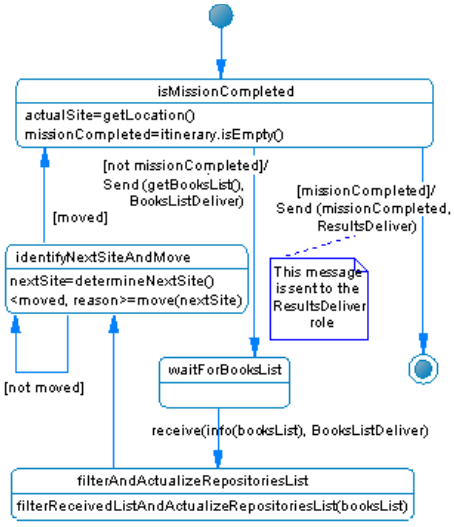


Fig. 16. searchBook task

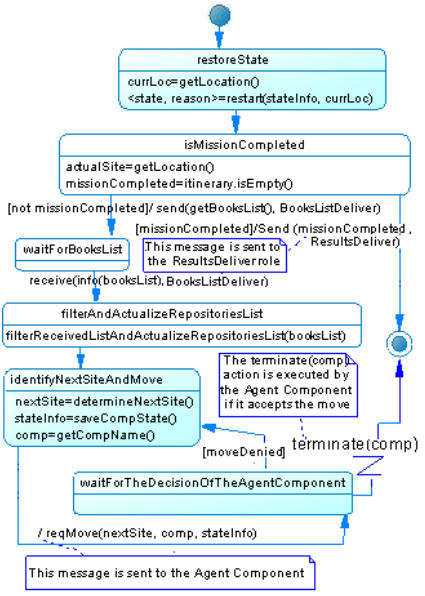


Fig. 17. searchBook mobile-component

Using m-Gaia, we identify in the *agent model* two types of agents: *MobileBookSeeker*$_m$ and *Librarian*, where the index (m) indicates that the agent is mobile. We also identify the following roles in the *role model*: *BooksListDeliver* (system role), *BookChecker* (interface role) and *ResultsDeliver* (user role). Fig. 18 shows the relationship between the roles and the agent types. In the *mobility model*, we distinguish two types of places: *mobilePlace* (with instance=1, to represent the laptop) and *stationaryPlace* (with instance=3, to represent site1, site2 and site3). *MobileBookSeeker*$_m$ can run on the two place types; where *Librarian* can run only on the *stationaryPlace* type. The *mobility model* allows to elaborate a *travel schema* for the mobile agent, which defines its origin place type (*stationaryPlace*: site1 for our example), its destination place type (*mobilePlace*: the laptop for our example) and a set of travel path (each one is a list of atomic movements). For our example, one travel path suffices. However, details about the syntax of atomic movements were not given in [34]: the authors have modeled their application example, realized it separately on Grasshopper, and then made manual correspondence between the modeled example and its realization.

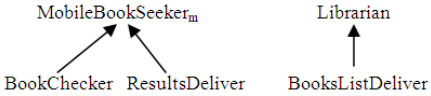


Fig. 18. Agent model for our example in m-Gaia

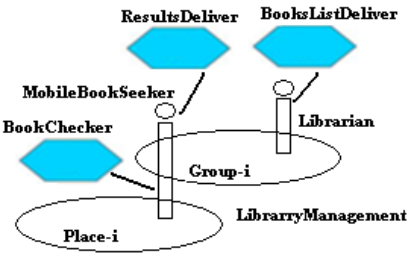


Fig. 19. Modeling our example using MAGR

```
(Mission (Name findBookRepositiories)
(Operation (Name searchBookRepositiories)
    (Action (MoveToPlace LibrarianSite1) (Name
    bookChecker) (Cmd (Name getBooksList)) (Cmd
    (Name booksFilter) (Args searchedBook)))

    (Action (MoveToPlace LibrarianSite2) (Name
    bookChecker) (Cmd (Name getBooksList)) (Cmd
    (Name booksFilter) (Args searchedBook)))

    (Action (MoveToPlace LibrarianSite3) (Name
    bookChecker) (Cmd (Name getBooksList)) (Cmd
    (Name booksFilter) (Args searchedBook)))
)
(Operation (Name deliverBookRepositiories)
    (Action (MoveToPlace clientAgency Laptop) (Name
    resultsDeliver) (Cmd (deliverBookRepositories)))
)
)
```

Fig. 20. MobileBookSeeker's mission

The MAGR's concepts (except *place* and *persistent role*) are the base of organization in our PIMM (see Fig. 5). Thus (organizational) models realized with MAGR are closer to ours. However, MAGR does not propose meta-models for agent, role, and Environment (i.e., domain). After elaborating the organizational model, it passes to the development step where it proposes MASL (Mobile Agent Script Language) to program MAS on Madkit (a mobile agent platform, supporting AGR and MAGR and compliant to MASIF and). With MASL, a mobile agent seems as executing a mission (representing its global goal). A mission is a set of operations (representing sub

goals). An operation is a set of actions (each one is a treatment executed on a different site). An action contains a move instruction and a set of commands (the finest elements of MASL). A possible modeling of our example using MAGR is presented in Fig. 19. On each site-i (i.e., site1, site2, site3 and the laptop), *LibraryManagement* is defined per a couple (Group-i, Place-i). Group-i contains non-persistent roles (*BooksListDeliver* and *ResulsDeliver*) and Place-i contains persistent roles (*BookCkecker*). *Librarian* is a stationary agent and thus can join only Group-i; however, *MobileBookSeeker* can join Place-i (and Group-i, as Place-i inherits from it). The script describing the itinerary and activity of *MobileBookSeeker* may be as in Fig. 20.

As shown, only MAGR and our PIMM consider organizational aspects (group, role) in the presence of mobility. On another side, m-Gaia and MAGR support the agent mobility by structuring its behavior as an itinerary which describes the task to do on each site; consequently, no effort is needed before or after moving. In contrast, the extended MaSE (respectively, our PIMM) allows for more flexibility in modeling the agent's behavior, and employs a *move* action (respectively, *Migration* action); however, an effort is needed before moving to save the states of the agent's components (respectively, to release roles and leave groups), and after moving to restore components (respectively, to eventually join groups and obtain roles). Table 1 summarizes the discussion on the studied methodologies extending MAS to support mobility. It interests only to the question of modeling mobility; for a comparison between MAS methodologies on others criteria see, for example, section 2.5 in [7], section 6 in [12] and section 6 in [15]).

Table 1. Mobility modeling in MAS methodologies

	before/ after migration's Treatment	Itinerary modeling	Mobile/ stationary agent distinction	Considering organizational aspects with mobility	Development process
Extended MaSE	yes by *Agent Component*	no	at level of components	No	RUP
m-Gaia	not needed	yes	yes	No	Cascade
MAGR	not needed	no (and yes at level of implementation)	yes	Yes	do not propose an elaborated process (*)
Our PIMM	yes	no	yes	Yes	MDE[11]
(*) The development cycle is quite limited. Gutknecht and Ferber have never wanted to propose a real process, in order to keep AGR generic and not reduce its potential of integration into ascendants or descendants processes. [17]					

[11] For details on MDA/MDE, see [19].

6 Conclusion

Our work contributes to bridge the gap between AOSE methodologies and mobile-agent systems. Indeed, we aim to propose an approach to develop multi-agents systems including mobile agents. In [20], we have presented our meta-model to design multi-agents systems including mobile agents and we have discussed it versus some formalisms extending UML for mobile-agents modeling. In this paper, we have discussed it versus particularly three works extending MAS methodologies (MaSE, Gaia, and AALAADIN) to support mobility. The PIMM was slightly updated (compared to its published version in [20]) to distinguish between mobile and stationary agents, to support flexible cloning and to treat the case where a mobile agent wants to move while holding (and eventually playing) several roles in parallel. We have discussed also the issue of agents' platforms compliance with MASIF and FIPA specifications.

As perspectives, we will first illustrate our MDE approach by transforming the PIM example built here into a PSM for JavAct (a mobile-agents platform we are using to experiment our approach), then into JavAct's code. After, it will be interesting to propose a PSMM for agents' platforms by examining works on the integration strategies combining both FIPA and MASIF standards. Then we need to complete our approach with a CIM meta-model and the transformation rules from CIM to PIM. Finally, it will be necessary to conduct experiments with real applications using different mobile-gents platforms (other than JavAct) to validate and enrich our approach.

References

1. Amor, M., Fuentes, L., Troya, J.M.: A component-based approach for interoperability across FIPA-compliant platforms. In: Klusch, M., Omicini, A., Ossowski, S., Laamanen, H. (eds.) CIA 2003. LNCS (LNAI), vol. 2782, pp. 266–280. Springer, Heidelberg (2003)
2. Amor, M., Fuentes, L., Vallecillo, A.: Bridging the Gap Between Agent-Oriented Design and Implementation Using MDA. In: AOSE, New York, pp. 93–108 (2004)
3. Aridor, Y., Lange, D.B.: Agent design patterns: elements of agent application design. In: AGENTS 1998, USA, pp. 108–115 (1998)
4. Bahri, M.R.: Une approche intégrée Mobile-UML/Réseaux de Pétri pour l'analyse des systèmes distribués à base d'agents mobiles. Doctoral thesis, University of Constantine, Algeria (2010)
5. Baumeister, H., Koch, N., Kosiuczenko, P., Wirsing, M.: Extending Activity Diagrams to Model Mobile Systems. Objects, Components, Architectures, Services, and Applications for a Networked World. In: International Conference NetObjectDays, NODe (2003)
6. Belloni, E., Marcos, C.: MAM-UML: an UML profile for the modeling of mobile-agent applications. In: The 24th SCCC, Arica, pp. 3–13 (2004)
7. Bernon, C., Gleizes, M.-P., Gauthier, P.: Méthodes orientées agent et multi-agent. In: Briot, J.-P. (ed.) Technologies des Systèmes Multi-Agents et Applications Industrielles, ch. 2, A. El Fallah-Seghrouchni,
8. Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J.J., Pavon, J., Gonzalez-Perez, C.: FAML: A Generic Metamodel for MAS Development. Journal of IEEE Transactions on Software Engineering 35(6), 841–863 (2009)

9. Blanc, X.: MDA en action: Ingénierie logicielle guidée par les modèles Ed. Eyrolles (2005)
10. Cao, J., Das, S.K.: Mobile Agents in Networking and Distributed Computing. Wiley Series in Agent Technology. John Wiley & Sons, Inc., USA (2012)
11. Cossentino, M., Bernon, C., Pavon, J.: Modeling and meta-modeling issues in agent oriented software engineering. The AgentLink AOSE TFG (2005)
12. Cossentino, M., Gaud, N., Hilaire, V., Galland, S., Koukam, A.: ASPECS: an Agent-oriented Software Process for Engineering Complex Systems, How to design agent societies under a holonic perspective. AAMAS 20(2), 260–304 (2009)
13. Da Silva, V.T., Noya, R.C., De Lucena, C.J.P.: Using the UML 2.0 Activity Diagram to Model Agent Plans and Actions. In: AAMAS 2005, pp. 594–600 (2005)
14. DeLoach, S.A.: Engineering Organization-Based Multiagent Systems. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds.) SELMAS 2005. LNCS, vol. 3914, pp. 109–125. Springer, Heidelberg (2006)
15. DeLoach, S.A., Garcia-Ojeda, J.C.: O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems. Int. Journal of AOSE 4(3), 244–280 (2010)
16. Demazeau, Y.: VOYELLES, HDR (Habilitation to Direct Research) thesis, INP Grenoble, France (2001)
17. Gauthier, P.: Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente. Doctoral thesis, University of Paul Sabatier, France (2004)
18. Gervais, M.-P., Muscutariu, F.: A UML Profile for MASIF Compliant Mobile Agent Platform. In: OMG's 2nd Workshop on UML for Enterprise Applications: Model Driven Solutions for the Enterprise, San Francisco, USA (2001)
19. Gherbi, T., Meslati, D., Borne, I.: MDE between Promises and Challenges. In: The 11th Int. Conf., Comp. Modeling & Simulation, UKSim 2009, Cambridge, pp. 152–155 (2009)
20. Gherbi, T., Borne, I., Meslati, D.: Un méta-modèle pour les applications basées sur les agents mobiles. In: CIEL 2012, Rennes, France, pp. 1–6 (2012)
21. Islam, N., Mallah, G.A., Shaikh, Z.A.: FIPA and MASIF standards: a comparative study and strategies for integration. In: National Software Engineering Conference, Rawalpindi, Pakistan (2010)
22. Jarraya, T.: Réutilisation des protocoles d'interaction et démarche orientée modèles pour le développement multi-agents. Doctoral thesis, University of Reims, France (2006)
23. Jarraya, T., Guessoum, Z.: Towards a model driven process for multi-agent system. In: Burkhard, H.-D., Lindemann, G., Verbrugge, R., Varga, L.Z. (eds.) CEEMAS 2007. LNCS (LNAI), vol. 4696, pp. 256–265. Springer, Heidelberg (2007)
24. Kusek, M., Jezic, G.: Modeling Agent Mobility with UML Sequence Diagram. In: AOSE, Ljubljana, Slovenia, pp. 51–63 (2005)
25. Loukil, A., Hachicha, H., Ghedira, K.: A proposed Approach to Model and to Implement Mobile Agents. IJCSNS 6(3B), 125–129 (2006)
26. Lima, E.F.A., Machado, P.D., Sampaio, F.R., Figueiredo, J.A.: An approach to modeling and applying mobile agent design patterns. In: ACM SIGSOFT, pp. 1–8 (2004)
27. Mansour, S., Ferber, J.: MAGR: Integrating mobility of agents with organizations. In: IADIS, Portugal (2007)
28. Mansour, S., Ferber, J.: Un modèle organisationnel pour les systèmes ouverts déployés à grande échelle. In: JFSMA 2007, Carcassonne, pp. 107–116 (2007)

29. Milojicic, D.: Mobile agent applications (trend wars). IEEE Concurrency 7(3), 80–90 (1999)
30. Picco, G.P., Murphy, A.L., Roman, G.C.: Lime: Linda Meets Mobility. In: ICSE 1999, pp. 368–377 (1999)
31. Rajguru, P.V.: Deshmukh. S. B.: Current trends and analysis of mobile agent application. In: Proceedings of NCETCT 2012, WJST, India, vol. 2(3), pp. 1–6 (2012)
32. Self, A., DeLoach, S.A.: Designing and Specifying Mobility within the Multiagent Systems Engineering Methodology. In: 18th ACM SAC, USA, pp. 50–55 (2003)
33. Spanoudakis, N., Moraitis, P.: Using ASEME methodology for model-driven agent systems development. In: AOSE Conf, Toronto., pp. 106–127 (2010)
34. Sutandiyo, W., Chetri, M.B., Loke, S.W., Krishnaswamy, S.: Extending the Gaia Methodology to Model Mobile Agent Systems. In: ICEIS, Porto, pp. 515–518 (2004)

A Fault Injection Based Approach to Assessment of Quality of Test Sets for BPEL Processes

Damian Grela, Krzysztof Sapiecha, and Joanna Strug

Department of Electrical and Computer Engineering,
Cracow University of Technology, Cracow, Poland
dgrela@pk.edu.pl, {pesapiec,pestrug}@cyf-kr.edu.pl

Abstract. Mutation testing is an effective technique for assessing a quality of test sets for software systems, but it suffers from high computational costs of generating and executing a large number of mutants. In the domain of BPEL processes each mutant needs to be deployed before it can be executed, thus the cost of processing mutants increases further. In contrast to mutation testing, fault injection is able to inject faults directly into the original process what reduces the redeployment requirement. The paper presents an experiment of the application of software fault injection to assess quality of test sets for BPEL processes. Faults are introduced by a Software Fault Injector for BPEL Processes (SFIBP). SFIBP simulates effects of the faults by modifying invocations of web-services and their internal variables. The experiment proved high superiority of the application of the SFIBP over the mutation testing, especially in the case of time requirements.

Keywords: Test Sets Quality Assessment, Mutation Testing, Fault Injection, Web-Services, Orchestration, Business Processes, BPEL.

1 Introduction

Recently, an application of WS-BPEL (Business Process Execution Language for Web-services) has become one of the most promising technologies for developing IT systems. WS-BPEL is a high level language that makes it possible to implement business processes as an orchestration of preexisting web-services [1]. A developer of an IT system should only select the most appropriate web-services and coordinate them, using WS-BPEL language, into business processes that cover specification requirements for the system. It leads to a very simple and structured architecture where only a special element of the process called its coordinator and communication links between the coordinator and the services need to be tested. Nevertheless, the testing should be performed with the help of a high quality test set to provide a confidence to system dependability. Thus, the development of tests should be supported by effective techniques for evaluating quality of test sets.

Mutation testing [2], [3], [4] is currently the most effective technique for quality evaluation of tests. In mutation testing faulty versions of an implementation of the system (so called mutants) are generated, by introducing small syntactic changes into the code, and executed against a test set. Although the technique is very efficient, it suffers from high computational cost of generating and executing mutants.

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 81–93, 2013.

In [5] a computational experiment aiming at evaluation of a novel approach that uses fault injection technique [6] to evaluate quality of tests for BPEL processes orchestrating web-services was presented. In contrast to mutation testing, fault injection can be performed at a run-time of the processes. Thus, it was shown that an application of this technique can significantly reduce the total cost of testing, as there is no need to create and compile a large number of the mutants. The work in [5] described an experiment that compared results of applying tests for mutants of a BPEL process with results of applying the same tests for the process but modified at a run-time by injecting faults. This paper extends the research presented in [5] by adding new examples and by providing general analysis of the deployment costs of BPEL processes. It also provides a short introduction of a possible improvement of the approach presented in [5].

The paper is organized as follows. Section 2 contains a brief description of the background and related work. The experiment and its results are described in Section 3. Section 4 outlines the way in which the approach presented in [5] can be improved. The paper ends with conclusions in section 5.

2 Background and Related Work

A number of papers related to different aspects of testing BPEL processes have already been published [7], [8], [9]. However, the papers do not consider the testing of BPEL processes in which the coordinator orchestrates web-services. A method of generation of test scenarios for validation of the coordinator of a BPEL process was given in [10]. Tests obtained by means of the method cover all functional requirements for the process and provide high validation accuracy [11]. Hence, such tests could also be used as a starting set of tests for the process.

Quality of generated test sets is an important issue, as only tests of high quality (high ability to detect faults) can help to provide dependable products [12], [13]. Several studies have proved validity of the mutation testing as a powerful technique for testing programs and for evaluation of the quality of test sets [13], [14]. For object systems tests are usually evaluated via mutation testing, but in case of BPEL processes this technique is very expensive due to the number of mutants that need to be generated, compiled, deployed and executed against the test set.

In mutation testing a quality of the test set is determined by a mutation score calculated as a ratio of mutants detected by the test set over all non-equivalent mutants. The higher is the mutation score the higher is the quality of tests. In [5] results of mutation testing were used as the reference when the results of fault injection were evaluated. The mutation testing is a white box testing technique that creates a large number of faulty programs (mutants) by introducing simple flaws (faults) in the original program. If a test case is able to detect the difference between the original program and the mutant, it is said that such test case kills the mutant. On the contrary, if none of the test cases is able to detect a difference, it is said that the mutant keeps alive for all used test cases. The mutants are created by applying so called mutation operators. Each of the mutation operators corresponds to a certain category of errors that the developer might commit. Such operators for various programming languages, including BPEL have already been designed [2], [3], [14].

Fault injection [6] is a popular technique that is mainly used for evaluation of fault-tolerance of computer systems. It consists in injection of deliberate faults into a running system and observation of its behavior. So called fault coverage [6] for a set of tests is measured. The fault coverage is expressed as a percentage of detected faults to all faults injected into the system. Fault coverage is used as a metric of quality of a set of tests and plays similar role as the mutation score for mutation testing.

Originally fault injection was applied to hardware systems, but currently it is also applied in software and mixed ones. Software fault injection (SFI) is implementation-oriented technique thus it targets computer applications and operating systems. SFI can be performed in near real-time, allowing for a large number of experiments. The technique was already applied for systems based on web services orchestration to emulate SOA faults at different levels [15], [16]. The approaches were built upon existing fault injection mechanisms. However, these solutions are still under development. It is not clear which types of SOA faults are supported, and how the faults are modeled and injected. Moreover, these works do not concern quality of test sets.

3 The Experiment

The main aim of the experiment presented in [5] was to provide evidences that for BPEL processes application of a software fault injection can evaluate quality of tests with similar accuracy as mutation testing does, but with lower time requirements than in case of mutation testing.

During the experiment, mutation testing and fault injection was applied to evaluate quality of the same sets of tests derived for 14 example BPEL processes orchestrating web-services (10 presented in [5] and 4 new). BPEL processes under consideration are described in Section 3.1, later on. Test sets, three for each of the processes, were provided.

The experiment consists of two main stages, each including several steps (Fig. 1):

1. application of traditional mutation testing, and
2. application of software fault injection.

In the first stage mutants of each of the original BPEL process were first generated, based on a set of mutation operators for BPEL (Section 3.2). Then each mutant was executed against a test set provided for the process and a mutation score for the set was calculated. The mutation score (MS) is expressed as follows:

$$MS(T) = \frac{M_K}{M_T - M_E} \cdot 100\% \text{ , where} \tag{1}$$

T - denotes a test set

M_K – is the number of mutants killed by the test set

M_T – is total number of generated mutants

M_E – is the number of equivalent mutants

In the second stage each original BPEL process was invoked and, while it was executed against its test set, different faults were randomly injected at run-time directly into the process and fault coverage for the set was calculated. The fault coverage (FC) is expressed as follows:

$$FC(T) = \frac{F_D}{F_I} \cdot 100\% \text{ ,where} \tag{2}$$

T - denotes a test set
F_D – is the number of faults detected by the test set,
F_I – is total number of injected faults.

The injected faults were generated from a set of fault injection operators (Section 3.2). The same test sets, as in the first stage, were used to stimulate the process.

Finally, results from applying mutation testing and fault injection were compared. Two criteria were considered during the comparison:

- the dependability of the fault injection approach in contrast to mutation testing (Section 3.3),
- the execution time of the fault injection approach in contrast to mutation testing (Section 3.4).

At each stage of the experiment the execution time required for evaluation of each test set for all processes were measured and compared.

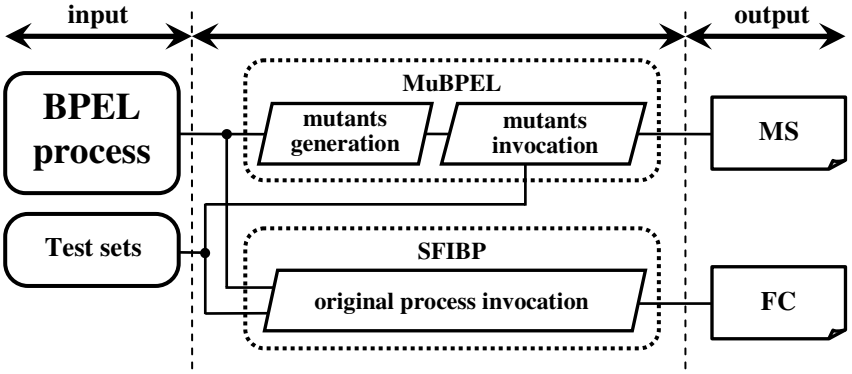


Fig. 1. Flow diagram of the experiment

The experiment was supported by applying two academic tools. In the first stage mutants were generated and executed with the help of MuBPEL [17]. In the second stage faults were introduced by Software Fault Injector for BPEL Processes (SFIBP) implemented by one of the authors.

3.1 Processes and Their Test Sets

The experiment was executed on 14 example BPEL processes (four our own and ten taken from a public repository shared by the University of Cadiz [19]). For each of the processes, three test sets were provided (first two randomly generated, last one obtained by checking paths method [11]. Table 1 contains the details: identifiers (ID), names and descriptions of the processes (Name and description), the number of

Table 1. BPEL processes used in the experiment

ID	Name and description	WS	TS
PDO	Planning Distribution of Orders helps its users to distribute orders among stores.	5	5/4/3
FRS	Football Reservation System allows its users to book tickets for football games, hotels to stay during the games and plane or train tickets to arrive at the games.	5	7/5/3
OB	Order Booking receives orders placed by users, it verifies the user and routes each order to two suppliers to get quotes and chooses the supplier that provided the lower quote.	8	12/10/4
PES	Project Evaluation System receives projects from the students, which are then evaluated by the teachers. On the basis of individual assessments of projects, system classifies student's final assessment.	6	9/14/8
LA1 **LA2**	Loan Approval concludes whether a certain request for a loan will be approved or not (it was published within the specification of the WS-BPEL 2.0 Standard. LoanApproval has two wariants.	2 2	6/9/5 6/9/5
SS	Squares Sum computes the value of sum (i=1 to n) for a certain value of n.	0	3/3/2
TS	Tac Service inverts the order of the lines in a file.	0	4/6/4
MP1 **MP2**	Market Place receives a price and offer from two partners: buyer and seller, and compares if the price offered by the buyer is equal or higher that set for the seller to sell. Market Place has two variants.	2 2	9/12/8 9/12/8
TI1 **TI2**	Trade Income models the behavior of managing a supermarket, controls the total profits that were generated until a certain date by the different establishments that the supermarket has, checks which section of the establishments has been more profitable, checks stock, controls whether some type of marketing is needed or whether the annual report must be done. TradeIncome has two variants.	7 7	12/6/6 12/6/6
MS1 **MS2**	Meta Search is a revised version of Philip Mayer's MetaSearch BPEL process, available at the BPELUnit site. MetaSearch implements a meta-search engine, which queries mockups of the Google and MSN search engines, interleaves their results and removes duplicates. MetaSearch has two variants that integrate results of the searches in different ways.	2 2	7/9/7 7/9/7

web-services used by the processes (WS) and the number of test cases provided in each of the test sets (TS). No fault tolerance mechanisms were used.

3.2 Mutation Operators and Fault Injection Operators

Mutation testing was performed with a help of MuBPEL [17]. The MuBPEL is a mutation testing tool for BPEL that automatically generates mutants of a BPEL process, executes the mutants against provided test set and finds the difference in output of both (mutated and original) BPEL processes. The MuBPEL generates mutants without the exclusion of the equivalent ones. This must be done by a user. A user also

needs to prepare a BPEL process and a set of its tests. The tests need to be created as test scripts using BPELUnit [18], which is an open-source WS-BPEL unit testing framework for BPEL processes.

In [14] a set of 26 mutation operators for BPEL processes was presented. In this experiment only 12 of them were used. The remaining 14 were skipped, as they refer to features of BPEL processes that are not supported by current version of SFIBP. All 12 operators listed in Table 2 have been implemented in the MuBPEL.

Table 2. Mutation operators used in the experiment

Operator	Description
	Identifier replacement operators
ISV	Replaces a variable identifier by another of the same type
	Expression operators
EAA	Replaces an arithmetic operator (+,-,*, div,mod) by another of the same type
EEU	Removes the unary minus operator from an expression
ERR	Replaces a relational operator (<,>,>=,<=,=,!=) by another of the same type
ELL	Replaces a logical operator (and,or) by another of the same type
ECC	Replaces a path operator (/,//) by another of the same type
ECN	Modifies a numerical constant incrementing or decrementing its value in one unit, adding or removing one digit
	Activity operators (concurrent)
ASF	Replaces a sequence activity by a flow activities
	Activity operators (non-concurrent)
AEL	Deletes an activity
AIE	Deletes an *elseif* element of the *else* element from an *if* activity
AWR	Replaces a *while* activity by *repeat-until* and vice versa
ASI	Exchanges the order of throw sequence child activities

Fault injection was executed with a help of a Software Fault Injector for BPEL Processes (SFIBP). The SFIBP is an execution-based injector [20], that is able to inject faults into the BPEL processes at a run-time, thus it simulates effects of the faults. Such approach helps to reduce costs of the experiment, as the faults are injected without changing the implementation of a process.

The SFIBP is implemented in Java as a special local web-service which acts as an intermediary between the BPEL process and the original web-service invoked by the process. Communication with the actual (used by the BPEL process) web-service is done with the help of build-in web-service client. The build-in client is able to remotely invoke any web-service, simultaneously modifying any parameter of the invocation. In this experiment each change introduced by SFIBP is based on one of a set of fault injection operators [21]. Table 3 presents the operators implemented in SFIBP.

Table 3. Fault injection operators used in the experiment

Identifier	Description
WS	Replaces requested web-service with another one
IP	Replaces values of a web-service input parameters
OP	Replaces values of a web-service output parameters
RV	Replaces a value of a variable

Configuration of the SFIBP includes setting of fault types, probability of their occurrence and of predefined web-services and values which are used when faults are injected. Information about the injected faults is stored in a log file. The total number of faults injected for a process always equals the number of mutants generated in the previous stage of the experiment.

3.3 Mutation Score and Fault Coverage - Results and Discussion

Table 4 gives the number of mutants generated for each BPEL process, mutants killed by each of the test sets and the mutation scores (*MS*) for each of the test sets.

Table 4. Results of mutation testing

BPEL process	mutants generated	mutants killed			MS [%]			
		TS1	TS2	TS3	TS1	TS2	TS3	average
PDO	229	184	186	190	80,34	81,22	82,97	81,51
FRS	219	193	182	191	88,13	83,11	87,21	86,15
OB	639	586	547	597	91,70	85,60	93,43	90,24
PES	687	492	552	596	71,61	80,34	86,75	79,57
LA1	28	20	23	23	71,43	82,14	82,14	78,57
LA2	36	30	32	33	83,33	88,89	91,67	87,96
SS	45	42	41	43	93,33	91,11	95,56	93,33
TS	53	43	47	48	81,13	88,67	90,56	86,79
MP1	29	25	27	27	86,21	93,10	93,10	90,80
MP2	45	39	42	44	86,66	93,33	97,78	92,59
TI1	557	551	528	556	98,92	94,79	99,82	97,85
TI2	615	403	419	436	65,52	68,13	70,89	68,13
MS1	525	411	471	479	78,28	87,81	91,24	85,77
MS2	554	383	395	409	69,13	71,30	73,83	71,88

Table 5 reports, for each of the BPEL processes, total numbers of faults injected, faults detected by each of the test sets and fault coverage (*FC*) for each of the test sets.

Table 5. Results of fault injection

BPEL process	Faults injected	Faults detected			FC [%]			
		TS1	TS2	TS3	TS1	TS2	TS3	average
PDO	229	179	181	188	78,16	79,04	82,09	79,77
FRS	219	187	177	190	85,39	80,82	86,76	84,32
OB	639	582	541	595	91,08	84,66	93,11	89,62
PES	687	486	547	591	70,74	79,62	86,03	78,80
LA1	28	20	22	23	71,43	78,57	82,14	77,38
LA2	36	28	31	33	77,78	86,11	91,66	85,19
SS	45	39	40	42	86,67	88,89	93,33	89,63
TS	53	42	45	48	79,24	84,90	90,57	84,90
MP1	29	18	21	23	64,28	75,00	82,14	73,81
MP2	45	39	41	43	86,66	91,11	95,56	91,11
TI1	557	546	521	553	98,02	93,53	99,28	96,94
TI2	615	408	422	443	66,34	68,62	72,03	68,99
MS1	525	405	456	474	77,14	86,86	90,29	84,76
MS2	554	378	386	412	68,23	69,68	74,37	70,76

Results of the fault injection are close to the results of the mutation testing for all evaluated test sets. As it can be observe in Table 4 and 5 an average fault coverage differs from an average mutation score from 0,62% (for OB) to 4,76% (for LA1). Higher consistency of results was observed in the case of larger systems. For such systems (OB, TI1, TI2, MS1 or MS2), the difference did not exceed 2%. The results obtained for the 4 newly added examples further confirm the results presented in the previous work [5].

Each technique uses its own fault model, thus changes made by mutation operators and faults injected by SFIBP are completely different kind of faults. Despite the lack of dependency between mutants and the injected faults, the results of both approaches are similar (the behavior of a process differs from the expected).

3.4 Execution Time - Results and Discussion

The most significant difference between the development of BPEL processes and the development of other types of software is the deployment. As far as mutation testing is concerned, a time of the deployment cannot be neglected, as each mutant needs to be deployed. Thus, the cost of deploying all mutants may significantly increase the total cost of applying mutation testing to BPEL processes.

A development of a BPEL process requires several steps to be performed. First, a coordinator of the process has to be implemented and compiled. The compilation of the processes includes mainly syntax checkout, and correctness of web-services invocation definition checkout. When the process passes the compilation, it is deployed. The deployment makes the process available to its users, as it places the process and its related files (the WSDL files for the process and web-services) onto a server where the process should reside.

An experiment was performed to see to what extend the deployment of mutants may affect the total time required for processing (i.e. deploying and executing) the mutants. The experiment was divided into two parts. The first part reflected the most pessimistic case of applying mutation testing where each mutant is deployed and executed against one test (although such case is not typical, it helped to determine the upper time limit). In this part several tests were performed, each consisted in deploying and executing a fix number of identical versions of an experimental BPEL process against one test. The tests invoked from 1 to 500 versions of the BPEL process, respectively and the total processing time was measured. The second part reflected application of fault injection where only the original process was deployed and then it was executed against tests several times. Thus, in this part each test consisted in deploying one BPEL process and executing it against a fix number of identical tests. Similarly to the first part the tests processed from 1 to 500 test sets, respectively and the total processing time was measured.

Fig. 2 shows results of the experiment. The dashed line shows the results obtained in the first part (mutation testing) and the continuous line represents results obtained in the second part (fault injection). As it can be seen on the diagram the time required for performing mutation testing is always higher than the time required for performing fault injection. Moreover, it grown also faster than in case of applying fault injection.

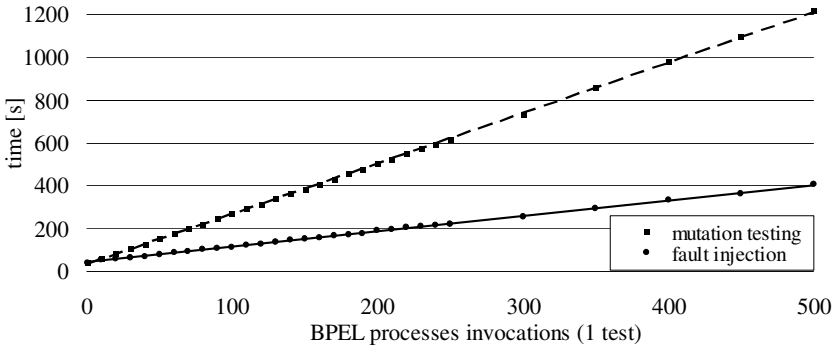


Fig. 2. BPEL processes deployment and execution time

Although the experiment reflected only the most pessimistic case of applying mutation testing, it was enough to observe the general trends and to see that the deployment time is a significant part of the total processing time. Thus, application of fault injection, instead of mutation testing, may speed up testing procedure.

The execution time recorded for the experimental BPEL processes confirm the above estimations. Table 6 presents, for each of the test sets, the exact execution time of mutation testing (MTt), fault injection (FIt) and it also shows the ratio of both execution times (MTt/FIt). All BPEL processes were executed on the same hardware and software configuration (Intel® Core™2 Duo 1.2GHz processor, 2GB RAM, Windows™ XP).

Table 6. Execution time of mutation testing and fault injection

BPEL proc.	Mutation Testing time (MTt) [s]			Fault Injection time (FIt) [s]			MTt/FIt			
	TS1	TS2	TS3	TS1	TS2	TS3	TS1	TS2	TS3	aver.
PDO	2311	1963	1444	1652	1304	941	1,40	1,50	1,53	1,479
FRS	1434	1176	918	937	679	493	1,53	1,73	1,86	1,708
OB	7517	6796	4049	4679	3959	2147	1,61	1,72	1,88	1,736
PES	17193	25490	16554	9874	14939	9547	1,74	1,71	1,73	1,725
LA1	239	309	214	194	245	175	1,23	1,26	1,22	1,239
LA2	1140	1482	1028	840	1071	766	1,36	1,38	1,34	1,361
SS	241	245	169	174	178	119	1,38	1,37	1,42	1,394
TS	474	667	481	359	548	364	1,32	1,22	1,32	1,286
MP1	1463	1995	1394	1238	1775	1174	1,18	1,12	1,19	1,164
MP2	1480	2004	1412	1218	1789	1159	1,22	1,12	1,22	1,184
TI1	59089	29128	29396	36932	16815	16932	1,59	1,73	1,74	1,689
TI2	37140	18326	18488	26181	11960	12042	1,41	1,53	1,54	1,495
MS1	11276	14418	11387	7583	9953	7606	1,49	1,45	1,50	1,478
MS2	6335	8089	6407	4406	5713	4410	1,43	1,41	1,45	1,436

Results presenting the execution time of fault injection in contrast to mutation testing for each of the test sets are additionally shown on Fig. 3. For each process the diagram shows that the time consumes by executing fault injection required from 53% to 89% of the time required for executing mutation testing.

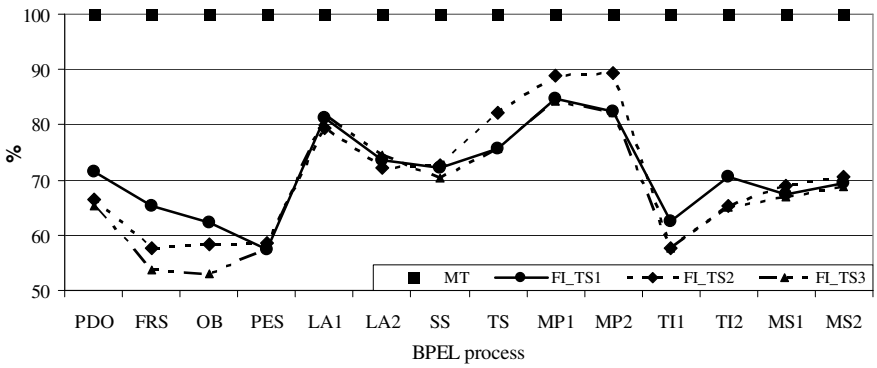


Fig. 3. Percentage comparison of execution times

So, the results proved that the fault injection is much faster than the mutation testing for all the test sets. It required on average about 65% of the time required by mutation based approach (the fault injection approach is about 1,5 times faster). Fault injection-based approach is particularly cost effective for large systems (e.g. OB, TI1, TI2) due to the lack of deployment of huge number of mutants. For smaller systems (e.g. MP1, MP2, LA1, LA2), the results are less effective thus for such systems the selection of test method is arbitrary.

4 Systematic Fault Injection Approach

The approach presented in this paper, though it assesses the test sets dependably, has some drawback. It required the user to set the number of fault injections before the evaluation started. Moreover, the best, most dependable results were obtained when their number was close to the number of generated mutants. It is of course problematic, especially if we would like to replace mutation based evaluation of tests quality by a fault injection based one.

However, detailed inspections of the results pointed out a possible solution to the problem. It has been observed that during random fault injection some faults were injected multiple times while others were not used at all. It suggests that a more systematic and controlled way of injecting faults may overcome some of the disadvantages.

In a systematic fault injection approach each possible fault is generated only once. A small experiment was carried out to check whether the systematic approach will may be beneficial with respect to the currently used random approach.

The PDO and OB processes were chosen for the experiment. Tables 7 and 8 show the execution time (Table 7) and fault coverage (Table 8) calculated for both processes by applying each of the approaches. To simplify the comparison between results obtained for these two examples by applying the random fault injection approach and the systematic one, the tables included also results presented earlier (in Tables 5 and 6).

Table 7. Execution time

BPEL proc.	Random Fault Injection time [RFIt] [s]			Systematic Fault Injection time (SFIt) [s]			RFIt/SFIt			
	TS1	TS2	TS3	TS1	TS2	TS3	TS1	TS2	TS3	aver.
PDO	1652	1304	941	541	448	304	3,05	2,91	3,09	3,019
OB	4679	3959	2147	1588	1282	693	2,95	3,09	3,10	3,044

Table 8. Fault coverage for random and systematic fault injection

Random Fault Injection								
BPEL process	Faults injected	Faults detected			FC [%]			
		TS1	TS2	TS3	TS1	TS2	TS3	Average
PDO	229	179	181	188	78,16	79,04	82,09	79,77
OB	639	582	541	595	91,08	84,66	93,11	89,62
Systematic Fault Injection								
BPEL process	Faults injected	Faults detected			FC [%]			
		TS1	TS2	TS3	TS1	TS2	TS3	Average
PDO	67	51	53	57	76,12	79,10	85,07	80,09
OB	169	152	146	157	89,94	86,39	92,90	89,74

Although two examples are not enough to reason about efficiency of the systematic approach, the results suggest that the systematic approach also provides accurate test evaluation results, but is about 3 times faster than the random approach.

5 Conclusions

Cost effective testing of extensive software systems requires specific approaches and different technologies adjusted to specific architectures. The experiment proved that testing based on SFI might be attractive for service oriented architectures (SOA) implemented with the help of BPEL. This is almost as effective as mutation testing but does not need elaboration of mutants. As the experiment on deployment time estimation shows, in case of mutation testing the deployment time affects the total execution time much heavier than in case of fault injection. So, for BPEL processes the fault injection approach is much faster because can be performed at a run-time of the process. Hence, this might be much more cost effective.

The experiment results show that even random testing detects a wide range of faults in the processes. Usually these faults are easy detectable ones. Using validation test sets seems to be more effective than random testing. The more complex is the process the higher are benefits from the fault injection and using validation test set, especially for time requirements. This last one is derived at the very beginning of the development of the system running the process, and thus does not need any extra effort while testing. The experiment proved that the random approach is an attractive alternative for mutation testing. However, the newly proposed systematic approach is also very promising. This approach will be further investigated in the future.

It is also interesting to see whether the fault injection based approach can be an alternative for mutation testing when other object oriented architectures are taken into account. In our future research more experiments on various types of SOA will be performed to strengthen the conclusions.

References

1. OASIS, Web Services Business Process Execution Language 2.0, Organization for the Advancement of Structured Information Standards (2007), http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
2. Offutt, A.J., Untch, R.H.: Mutation testing for the new century. In: Mutation, Uniting the Orthogonal, pp. 34–44. Kluwer Academic Publishers, Norwell (2001)
3. Woodward, M.R.: Mutation testing — its origin and evolution. Information and Software Technology 35(3), 163–169 (1993)
4. Strug, J., Strug, B.: Machine Learning Approach in Mutation Testing. In: Nielsen, B., Weise, C. (eds.) ICTSS 2012. LNCS, vol. 7641, pp. 200–214. Springer, Heidelberg (2012)
5. Grela, D., Sapiecha, K., Strug, J.: An application of software fault injection for assessment of quality of test sets for business processes orchestrating web-services. In: Proceedings of the 8th International Conference On Evaluation of Novel Approaches to Software Engineering (ENASE 2013), Angers, France, pp. 56–62 (2013)
6. Hsueh, M.C., Tsai, T.K., Iyer, R.K.: Fault Injection Techniques and Tools. IEEE Computer 30(4), 75–82 (1997)
7. Dong, W.-L., Yu, H., Zhang, Y.-B.: Testing BPEL-based web service composition using high-level Petri nets. In: EDOC 2006: Tenth IEEE International Enterprise Distributed Object Computing Conference, Hong Kong, China (2006)
8. Yan, J., Li, Z., Yuan, Y., Sun, W., Zhang, J.: BPEL4WS unit testing: Test case generation using a concurrent path analysis approach. In: ISSRE 2006: 17th International Symposium on Software Reliability Engineering, Raleigh, North Carolina, USA, pp. 75–84 (2006)
9. Yuan, Y., Li, Z., Sun, W.: A graph-search based approach to BPEL4WS test generation. In: ICSEA 2006: International Conference on Software Engineering Advances, Papeete, Tahiti, French Polynesia, p. 14 (2006)
10. Sapiecha, K., Grela, D.: Test scenarios generation for certain class of processes defined in BPEL language. In: Annales UMCS - Informatica, vol. 8(2), pp. 75–87 (2008)
11. Sapiecha, K., Grela, D.: Automating test case generation for requirements specification for processes orchestrating web services. In: Information Systems Analysis and Specification, Barcelona, Spain. 10th International Conference on Enterprise Information Systems (ICEIS), vol. 1, pp. 381–384 (2008)
12. Wagner, S., Gericke, J., Wiemann, M.: Multi-Dimensional Measures for Test Case Quality. In: IEEE International Conference on Software Testing Verification and Validation Workshop, ICSTW 2008 (2008)
13. Farooq, U., Lam, C.P.: Evolving the Quality of a Model Based Test Suite. In: International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2009 (2009)
14. Estero-Botaro, A., Palomo-Lozano, F., Medina-Bulo, I.: Mutation operators for WS-BPEL 2.0. In: ICSSEA 2008: 21th International Conference on Software & Systems Engineering and their Applications, Paris, France (2008)

15. Reinecke, P., Wolter, K.: Towards a multi-level fault injection test-bed for service-oriented architectures - requirements for parameterisations. In: 27th International Symposium on Reliable Distributed Systems, Napoli, Italy (2008)
16. Juszczyk, L., Dustdar, S.: Programmable fault injection test-beds for complex SOA. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 411–425. Springer, Heidelberg (2010)
17. MuBPEL - WS-BPEL Testing Tools, http://neptuno.uca.es/redmine/projects/sources-fm/wiki/MuBPEL
18. Mayer, P., Lubke, D.: Towards a BPEL unit testing framework. In: Proceedings of the workshop on Testing, analysis and verification of web services and applications, TAV-WEB 2006, pp. 33–42. ACM, New York (2006)
19. University of Cadiz WS-BPEL Composition Repository, http://neptuno.uca.es/redmine/projects/wsbpel-comp-repo
20. Benso, A., Prinetto, P.: Fault injection techniques and tools for embedded systems reliability evaluation. Kluwer Academic Publishers, Holland (2003)
21. Durães, J., Madeira, H.: Software Faults - A field data Study and a practical approach. Trans. of Software Engineering (2006)

Comparing Two Class Composition Approaches

Fernando Barbosa[1] and Ademar Aguiar[2]

[1] Escola Superior de Tecnologia, Instituto Politécnico de Castelo Branco,
Av. do Empresário, Castelo Branco, Portugal
fsergio@ipcb.pt
[2] INESC TEC and Faculdade de Engenharia da Universidade do Porto,
Rua Dr. Roberto Frias, Porto, Portugal
ademar.aguiar@fe.up.pt

Abstract. The presence of code replication can be a consequence of a lack in the composition mechanisms where classes are insufficient to reuse the code that is replicated. To extend the reuse of pieces of code some proposals have been made that try to compose classes using those pieces of code. In this paper we compare two of those approaches: Traits and Roles. We compare their compositions mechanisms and how we can use them to reduce code replication. To study the extent to which they reduce code replication we conducted a case study using the JHotDraw framework where we detect and remove code replication using each technique. Results from the case study show that roles have an advantage over traits, as they are capable of removing more code replication.

Keywords: Roles, Traits, Code Reuse, Modularity, Composition, Inheritance.

1 Introduction

Code clones, identical blocks of code, are a hint that the system needs to be refactored [1]. However code clones appear in most systems, specially in large ones [2,3]. Code clones impair maintenance and evolution of a system [3]. One problem is the inconsistence in updating, where a bug in a code block is propagated to all its clones, and is fixed in most but not all occurrences. Code clones also have negative effects in program evolution, comprehensibility and cost [4].

One origin of clones is the lack of composition mechanisms [2,3,4]. This makes it harder to deal with crosscutting concerns - concerns that a class must deal with but are not its main concern. When dealing with the same concern classes tend to use similar code. This is more frequent in languages without multiple inheritance, but multiple inheritance has so many practical problems that it has been left out of recent languages, like Java and C#.

Some clones could be avoided if a language had other composition mechanisms. Several proposals are available, like multiple inheritance, mixins [5], traits [6,7], features [8] and aspects [9].

Traits can be seen as a set of methods that provide common behavior. When a class uses a trait its methods are added to the class. The class also provides glue code to compose the several traits. Traits cannot store state. State is maintained by the class that uses the trait.

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 94–109, 2013.

When a class plays a role the role methods are added to the class interface. Thus an object's behavior is defined by the composition of all roles its class declares to play. A class can configure the role to its needs by configuring types and methods names. Roles support state and visibility control.

Composing classes using traits or roles can minimize the code replication due to limitations of the composition mechanism. To assess this we conducted an experiment to account how both approaches could be used to remove the replicated code found in the JHotDraw Framework. We briefly present the two approaches then compare them showing how they deal with conflict resolution, composition order, etc.

We identified code clones using a clone detecting tool, and grouped them according to their concerns. We then tried to develop a role and a trait for each concern, thus removing the clones. We developed roles for nearly all detected concerns, but couldn't do the same for traits.

We can summarize our paper contributions as: a comparison of roles and traits features, ways of reducing replicated code using traits and roles; a comparison of how roles and traits tackle the problem of reducing duplicated code and identifying which clones they can eliminate; a case study showing how each approach reduces replicated code in an open source system.

This paper is organized as follows. Section 2 presents Traits and Section 3 presents roles. In Section 4 we compare the two approaches. Section 5 shows how to remove clones using roles and using traits and section 6 presents the JHotDraw framework case study. Related work is presented in section 7 and section 8 concludes the paper.

2 Traits in a Nutshell

Traits are units of code reuse and a class can be constructed using several traits [6,7]. Traits have a flattening property: a class can be seen indifferently as a collection of methods or as composed by traits. The fact that the class can be seen as a whole promotes understanding and the fact that it can be composed promotes reuse.

In Traits a class can be constructed by using inheritance and by adding traits. The class must supply all state variables and glue code. The glue code is the set of methods that the trait requires the class to provide (for example, accessor methods for the state variables). Thus a class can be decomposed into a set of coherent features and the glue code connects the various features together.

According to [6], Traits have the following properties:

- A trait provides methods that implement behavior
- A trait requires a set of methods that serve as parameters for the provided behavior.
- Traits do not specify state variables, and methods provided by traits never access state variables.
- Classes and traits can be composed from traits.
- The composition order of traits is irrelevant.
- Conflicting methods must be explicitly resolved.
- Trait composition does not affect the semantics of a class: the meaning of the class is the same as it would be if all of the methods obtained from the trait(s) were defined directly in the class.
- Similarly, trait composition does not affect the semantics of a trait.

A class can redefine its superclass's and its trait's methods. Conflicts arise when unrelated traits have methods with the same signature. The conflict must be solved explicitly by redefining the conflicting method in the class. The conflict is thus resolved locally. To access the conflicting methods Traits support aliases. It works by giving an alias to a method so it can be used in the class without trouble. To prevent conflicts from occurring in the first place traits also support the exclusion of methods.

Some attempts to bring traits into Java-like languages have been made [10,11]. To compare the Trait approach to the Role approach we used Chai [11]. Chai is an extension to the Java language and so is our JavaStage language, so we can argue that the differences between the final code is due integrally to each approach and not to the underlying language. The traits examples in this paper are presented using the Chai syntax and derive from the example shown in [11].

Figure 1 shows trait declaration in Chai and its use by classes. We can see requirement of methods in the TEmptyCircle: it offers a draw method and requires the class to provide the drawPoint and getRadius, with the specified signature. The same methods are also required by TFilledCircle. The code also shows a Circle class, representing a circle, and two subclasses composed by traits and that inherit from Circle. The ScreenEmptyCircle class is an empty circle that can be drawn in the Screen, so it uses TEmptyCircle and TScreenShape. The methods required by TEmptyCircle are supplied by Circle and TScreenShape, so ScreenEmptyCircle does not need to provide them itself. PrintedFilledCircle is a filled circle than can be printed in a printer, so it inherits from Circle and uses TFilledCircle and TPrintedShape. TFilledCircle required methods are supplied by Circle and TPrintedShape. In the TPrintedShaped case the class needed to alias the trait method for the required name.

For more information on Traits we refer to [6,7] and for Chai we refer to [11].

```
class Circle {
  int radius;
  int getRadius() { ... }
}
trait TEmptyCircle {
  requires {
    void drawPoint(int x, int y);
    int getRadius();
  }
  void draw() { ... }
}
trait TFilledCircle {
  requires {
    void drawPoint(int x, int y);
    int getRadius();
  }
  void draw() { ... }
}
```

```
trait TScreenShape {
  void drawPoint(int x, int y) {...}
}
trait TPrintedShape {
  void printPoint(int x, int y){...}
}
class ScreenEmptyCircle
  extends Circle
  uses TEmptyCircle,TScreenShape { }
class PrintedFilledCircle
  extends Circle
  uses TFilledCircle,TPrintedShape {
     alias {
       void printPoint(int x, int y)
       from TPrintedShape as
       void drawPoint(int x, int y)
     }
}
```

Fig. 1. Trait example (adapted from (Smith, 2005)

3 Roles in a Nutshell

We use roles as a basic construct from which we can compose classes. Roles provide the basic behaviour for concerns that the classes must deal with but are not their main

concern. Thus we can better modularize the construction of classes. We must mention that we use roles statically as defined by Riehle in [12] where he uses them as static entities for modelling purposes. We do not use roles as dynamic entities that can be attached or detached from an object at runtime. Since there is much work on the use of dynamic roles [13,14,15] this must be mentioned to avoid confusion.

To program with roles we use JavaStage, an extension to Java [16]. Examples in this paper use the JavaStage syntax. Figure 2 shows the role version of the trait example of Figure 1.

A role may define methods and fields including access levels. A class can play any number of roles, and can even play the same role more than once. A class playing a role is a player of that role. When a class plays a role all the non private methods of the role are added to the class. To play a role the class uses a plays directive and gives the role an identity. To refer to the role the class uses its identity. Roles can inherit from roles and can also play other roles.

A role may require the player to have specific methods. Those methods are stated in a requirement list, which indicates who must supply the method and the method signature. The Performer keyword indicates that the supplier is the player. Performer is used within a role as a place-holder for the player's type. This enables roles to declare fields and parameters of the type of the player.

JavaStage has a method renaming mechanism that allows the renaming of methods with a simple configuration. Each name may have three parts: a configurable one and two fixed. Both fixed parts are optional. The configurable part is bounded by #, like in the example: fixed#configurable#fixed.

The name configuration is done by the class playing the role in the plays clause. To play the role the class must define all configurable methods.

```
class Circle {
  int radius;
  int getRadius() { ... }
}
role EmptyCircle {
  requires Performer implements
     void #draw#(int x, int y);
  requires Performer implements
     int getRadius();
  void draw() { ... }
}
role FilledCircle {
  requires Performer implements
     void #draw#(int x, int y);
  requires Performer implements
     int getRadius();
  void draw() { ... }
}
```

```
role ScreenShape {
  void drawPoint(int x,int y){ ... }
}
role PrintedShape {
  void printPoint(int x,int y){ ... }
}
class ScreenEmptyCircle
 extends Circle {
   plays EmptyCircle( draw= drawPoint
         ) emptyCircle;
  plays ScreenShape screenShp:
}
class PrintedFilledCircle
 extends Circle {
   plays FilledCircle(
     draw = printPoint ) fillCircle;
  plays PrintedShape;
}
```

Fig. 2. Role example, equivalent to the traits' example in Figure 1

It's possible to declare several versions of a method using multiple definitions of the configurable name. This way, methods with the same structure are defined once.

Role members have all the visibility control available to classes and a protected role member is accessible to its players and subroles. A protected class member is

also accessible to roles. A class can reduce the visibility of the role members. If a class uses protected in the plays clause then all the public role methods are imported to the class as protected.

Class defined methods always take precedence over role methods and role methods take precedence over inherited methods. Conflicts may arise when a class plays roles that have methods with the same signature. When conflicts arise the compiler issues a warning. Developers can handle the conflict by redefining that method and calling the intended method. This is not mandatory because the compiler uses, by default, the method of the first role in the plays clause order.

JavaStage supports role constructors but does not allow direct role instantiation. For more information on roles and JavaStage we refer to [16].

4 A Comparison between Roles and Traits

For comparing roles and traits we follow a few key points that both approaches must deal with and describe how each handled the situation.

Unit of Composition. In roles the unit of composition is the role while in traits it is the trait.

Inheritance. Roles and traits are targeted for single inheritance languages so there is no multiple inheritance support. Roles can play other roles and traits can use other traits. Both approaches also support a class using the same unit several times. In a class, to access the features of the superclass both approaches use the super keyword. In a role, however, the super keyword refers to the super role, as roles can inherit from other roles. In a trait it refers to the superclass of the composing class.

State Support. Roles can have state and it does not cause any conflict because to access role state the class must use the role identity thus no conflicts arise. Traits do not support state. Proposals to solve this introduced a significant complexity to the trait model and encapsulation problems [17]. When modelling a concept we, often, need to express state. For example, to model a container we need a structure for storage. Forcing the composing class to supply that structure is rather breaking the container's encapsulation.

Conflict Resolution. Both approaches follow the same rules for method overriding. The class overrides methods from roles/traits and roles/traits override the class inherited methods. Conflicts may arise when methods with the same signature are provided by more than one unit. In traits the conflict must be resolved explicitly while in roles the method of the first played role is used (there is a compiler warning). In both cases it is the class composer that decides which method to use. In traits he can choose to exclude some methods so there is no conflict or he can redefine the method and use aliases to refer to each of the conflicting methods. In roles there is no exclusion and the class composer must redefine the conflicting method if he wishes to override the rule of using the method of the first role.

Composition Order. The order in which traits are composed is symmetric so order of composition is irrelevant. The same applies for the roles when there are no conflicting methods. When there are conflicting methods the order of the plays will dictate which method is used. This, however, is not mandatory as discussed previously.

Method Renaming vs. Aliases. There is a fundamental difference between aliases in traits and method renaming in the roles. The traits aliases are used only by the class for distinguishing conflicting methods, the class interface is not affected. In roles the renaming affects the class interface. This means that a class may be able to tailor its interface to suit its needs and not be limited by the role interface. The renaming mechanism of the roles also allows renaming several methods in one go, while aliases in traits are made one by one. Roles renaming scheme can provide multiple versions of a method. Traits aliases can be applied to any method, while on roles only the configurable methods can be configured.

Flat and Composite View. Both approaches support a flat view of the class as well as a composite view. Thus a class can be seen as a set of methods, the flat view, or as being composed by several units of composition, the composite view. The class interface in both views is exactly the same. The main difference between the two is that a trait method is seen just like a class method, and a role method is always a role method and each reference to other methods will always refer to role methods. For example, suppose a trait that defines the methods foo and bar, where bar calls foo. If the class overrides the foo method then the trait bar method will call the foo method on the class not on the trait. The same situation is handled differently by roles. If the method bar of the role is called then it will call the foo method on the role and not on the class. For a role method to call a class method it must do it explicitly using the perfomer keyword.

Visibility Control. Traits have no visibility control. Freezable traits [18] compensate this by allowing classes to freeze/unfreeze methods, i.e., declare a method as private (freeze) or making it public (defrost). But there is no way to express access constraints between class and trait. For example, fields should be accessed directly only by the owner's code. Traits do no support this. Roles on the other hand support all Java access levels, so a specific interface between role and class is possible.

Stating Requirements. The use of generic types is a useful feature in most languages, especially for dealing with object collections. Traits can require methods from the class that uses them, but cannot impose restriction on generic types it interacts with. The requires statement of roles indicates the method signature and which type it is required from. This allows roles not only to require methods from the class but also from other collaborators types.

5 Removing Clones

We want to assess if the extra units of composition roles and traits provide are capable of reducing code clones. To remove code clones refactorings [1] are normally used.

The ones most used for removing code clones are: Extract Method; Pull Up Method, Pull Up Field, Extract Superclass, Extract Class and Form Template Method [19, 20, 21].

We identified three clone types where roles or traits can be applied to remove code clones that fall outside the scope of these refactorings or produce better results. The clone types all have method granularity, so if actual clones do not have method granularity other refactorings must be used. Clone types are: Clones with identical code; Clones with similar code but using different types; Clones with similar code but using different method names with or without different types.

Clones with Identical Code. These clones have identical methods and/or fields. This could be handled by the Extract Class refactory, but we argue that this is one situation where roles/traits produce better results. Extract Class forces the original class to provide delegate methods to the newly created class. With roles and traits those methods are not required. We put the code in the role/trait and then compose the class using it.

The application of roles and traits is shown in Figure 3. The figure represents two classes with replicated code. Both classes have different, unrelated, superclasses so Pull Up Method and Extract Superclass cannot be used. The replicated code was placed in a trait and a role.

Both solutions are similar, the difference is that roles support state so they do not require the getX and setX methods and can even provide them. Traits require the class to supply those methods.

Clones with Similar Code but Using Different Types. These could be handled by Extract Class, using type parameters. For example we can build a Company class that manages workers and we can build a PolyLine that stores points. Both classes will have code for adding and removing workers/points, so there will be replicated code between them, only the stored type is different. We can create a unit responsible for this management. We show this example in Figure 4. For simplicity and space we used arrays and do not show the management code.

```
trait TContainer<T>{
  requires { T[] getAll(); }
  void add(T t){...}
  void remove( T t){ ... }
}
class Company uses TContainer<Worker>{
  Worker arr[];
  Worker[] getAll(){ return arr; }
}
class PolyLine uses TContainer<Point>{
  Point arr[];
  Point[] getAll() { return arr; }
}
```

```
role Container<T> {
  private T arr[];
  void add( T t ){...}
  void remove( T t){...}
  T[] getAll() { return arr; }
}
class Company {
 plays Container<Worker>cWorker;
}
class PolyLine {
 plays Container<Point> cPoint;
}
```

Fig. 3. Removing identical clones with different types using roles and traits

```
class A extends SuperA {
  private int x;
  int getX() { return x; }
  void setX(int x){ this.x = x; }
  void foo( ) {  // more code
    x += 14;
  }
  void bar() { ... }
}
```

```
class B extends SuperB {
  private int x;
  int getX() { return x; }
  void setX(int x){ this.x = x;}
  void foo( ) { // more code
    x += 14;
  }
  void bar() { ... }
}
```

```
trait TOne {
  requires{ int getX();
            void setX( int x );}
  void foo( ) { // more code
    setX( getX() + 14 );
  }
  void bar() { ... }
}
class A extends SuperA uses Tone {
  private int x;
  int getX() { return x; }
  void setX( int x){ this.x = x; }
}
class B extends SuperB uses Tone {
  private int x;
  int getX() { return x; }
  void setX( int x){ this.x = x; }
}
```

```
role ROne {
  private int x;
  int getX() { return x; }
  void setX(int x){ this.x = x; }
  void foo( ){ // more code
    x += 14;
  }
  void bar() { ... }
}
class A extends SuperA {
  plays ROne r1;
}
class B extends SuperB {
  plays ROne r1;
}
```

Fig. 4. Removing identical clones using roles and traits

There is a limitation in traits that may render a solution impossible: traits cannot require methods from other sources other than the class that uses them. A possible example is the Observer pattern [22], where subjects maintain a list of observers and notify them when changes occur using an update method. The observer management is similar to the container problem just described, so the same solution can be applied. The problem lies in calling the update method. For calling this method the Trait must specify the type of the observer otherwise it cannot call a method on it. Roles can solve this by requiring the Observer type to implement an update method, as shown in Figure 5, where a Figure class notifies FigureObservers whenever it is changed. The solution reuses the container role and just adds the notify method.

```
role Subject<T> extends Container<T>{
  requires T implements void update();
  void notify( ) {
    for( T t : getAll() ) t.update();
  }
}
class Figure {
  plays Subject<FigureObserver> figSubject;
}
```

Fig. 5. Defining requirements on collaborators types

Clones with Similar Code but Using Different Method Names with or without Different Types. These clones have identical code but the names of the methods are

not identical. The used types may also be different. For example we could change the Company and Polyline example and change them so that each had different names. The company would have addWorker and removeWorker methods while PolyLine would have addPoint and removePoint.

Traits aliases do not cope with these changes as they only affect the methods internally. With traits we would have to uniform the methods names and then apply the previous topic solution. We show how this situation is handled by roles in Figure 6. The Company class also shows how we can use the multiple method versions to produce an addWorker and an addEmployee method.

```
role Container<T> {
  private T arr[];
  void add#Thing#( T t ) { ... }
  void remove#Thing#( T t ) {...}
}
class Company {
  plays Container<Worker>( Thing = Worker, Thing = Employee ) cWorker;
}
class PolyLine {
  plays Container<Point>( Thing = Point ) cPoint;
}
```

Fig. 6. Removing identical clones with different methods and types using roles

6 Case Study

To compare how roles and traits are capable of reducing code replication we applied both to the JHotDraw framework. The framework defines the basic structure for a GUI based editor with tools, different views, user-defined graphical figures, etc.

6.1 Case Study Setup

We searched for replicated code with CCFinderX [23], a clone detection tool used in aspect mining works [24]. We filtered clones inside the same file (same class), thus eliminating clones that could use Extract Method or similar. We want to assess traits and roles capability to reduce the code clones derived from compositional limitations, so we only want clones that are not removable with traditional refactorings.

The first result included 271 clones, reduced to 146 after filtering. These were manually inspected. 41 false clones were removed leaving a final 105 sets. Some clones only had similar structures, but as they focused on the same concern we did consider them. This will explain some unresolved concerns.

We grouped clones according to their concerns. This helped us decide which role/trait to develop. We identified 42 concerns, but 5 were removed (2 could be easily refactored, 1 was deprecated code and 2 were classes pending substitution).

We've decided not to change any class interface or any concern implementation, so the framework is unchanged. This can restrict roles/traits development but we want to assess how we can reduce code replication, not redesign the framework. Roles were developed and compiled with JavaStage [16] and traits developed with Chai [11].

Results are shown on table 1. For each concern it shows how many clones were associated and how many classes were affected. It also shows the number of lines of code (LOC) that the clone had, the lines of code that were used by Roles and Traits, and the ratio between the various solutions.

LOC are a good measure on the effort that each approach requires, because both use Java as the underlying language, the syntax of both solutions is analogous and we made an effort to uniform the LOC count. We counted as LOC the requirements statements that traits and roles use. We also counted as LOC the roles' plays directive and the traits' uses directive. This overhead can lead a small clone to have more LOC in the solution than in the original form but the fact that there is no clone gives the system a great modularity advantage.

In the cases where roles removed clones and traits did not, the table shows the roles features that allowed them to remove the clone. For the concerns that neither technique worked it states the reason why they failed.

6.2 Results Analysis

Table 1 shows that from the 38 concerns only 8 (21%) concerns were not resolved with roles. Traits failed to resolve 15 (39%) concerns. It also shows that traits were not able to resolve the clones that roles could not. The final outcome is better than these numbers indicate as we will discuss.

We can see from table 1 that roles never had worst results than traits, succeeding in 7 concerns where traits failed and fared better in 13 more concerns. This indicates that roles are, at least, as good as traits in reducing replicated code.

Concerns Resolved with Roles and Traits. Comparing the LOC ratio of both approaches, in those concerns both resolved, one finds that in average roles only have 83% of the traits code and 68% of the original code, so the effort of developing the role system seems smaller.

In 6 concerns we were able to reuse roles from the role library developed in [25]. From those, 3 are solvable with traits but we had to develop a special trait for each concern and could not reuse them from a library. This explains the great difference in LOC in these concerns.

Supporting state is the role feature responsible for the fewer code used in roles. The class instead of having to declare each field and provide getters and setters would have the field and methods defined in the role. This is no small advantage not only in LOC but also in terms of abstraction and encapsulation.

The multiple method version also enabled roles to have less code, because a single method definition can provide several methods.

Concerns Resolved by Roles only. Roles were able to resolve 7 concerns that traits did not. From these, 3 used roles from the library. Requiring and renaming methods from other participants is the feature that enables roles to solve more clones.

From all the concerns roles resolved, two exhibit a higher LOC than the original implementation: "Handle creation" and the "Polygon locator".

Table 1. Identified concerns with the number of associated clones and affected classes. It also shows the LOC for each approach and respective ratios.

	Concern	clone #	class #	Original LOC	Roles LOC	Roles/ Original	Traits LOC	Roles/ Traits	Traits/ Original
Resolved by Roles and Traits	Drawing Handles	8	15	64	40	63%	40	100%	63%
	Setting up the undo activity before executing a Command	2	8	56	44	79%	44	100%	79%
	BringToFront/SendToBack Commands	1	2	20	12	60%	12	100%	60%
	Handle creation	11	20	70	87	124%	87	100%	124%
	Drawing polygons	1	2	12	11	92%	11	100%	92%
	Palette Listener	1	2	20	17	85%	17	100%	85%
	DisplayBox persistence	2	5	35	12	34%	12	100%	34%
	DisplayBox handling	6	8	58	29	50%	60	48%	103%
	DesktopListener Subject	2	3	63	45	71%	55	82%	87%
	Changing connections	3	3	98	53	54%	65	82%	66%
	Finding connectable figure	1	3	98	53	54%	65	82%	66%
	Testing command executability	5	7	14	14	100%	15	93%	107%
	Floating text holder	2	2	47	36	77%	47	77%	100%
	DrawingViewListener Subject	2	4	63	26*	41%	47	55%	75%
	Setting text in a text Figure	2	2	36	22	61%	32	69%	89%
	Enumerator	1	3	33	11*	33%	37	30%	112%
	Figure Listener that resends notifications	2	3	35	23*	66%	37	62%	106%
	Menu enabling	1	2	20	14	70%	14	100%	70%
	Version control	1	2	12	9	75%	9	100%	75%
	Selected button manager	1	2	18	12	67%	16	75%	89%
	Text attributes management	2	2	206	120	58%	149	81%	72%
	Updating DrawingView Strategy	1	2	29	26	90%	32	81%	110%
	Connection insets computing	1	3	10	7	70%	7	100%	70%
Resolved only by Roles							**Roles features**		
	Undo/Redo Commands	1	2	32	31	97%	br rp g		
	Changing connection handles	1	2	20	19	95%	br rp g		
	Polygon and PolyLine Handles	3	2	32	28	88%	rp		
	Tools and Commands Dispatchers	6	4	89	32*	36%	br rp		
	Figure/Handle and Enumerator	1	2	33	2*	6%	br rp		
	Polygon locator	1	2	13	20	154%	rp		
	Drawing editor	1	3	54	28*	52%	mv br rp		
Unresolved				**Reason**					
	Desktop initial configurations	1	2	required too much configuration					
	Persistence (read/write)	3	6	similar but not quite identical code					
	UndoActivity	13	24	Undoactivity inner classes constructors					
	Creating UndoActivity	14	18	after other roles was just a line of code					
	Handle manipulation starting action	3	5	required too much configuration					
	Point is inside Figure	3	6	code too small					
	DrawingView Listener	1	2	perfomance issues					
	Mouse motion handling	1	2	code too small					

*= reused role from library, br= block renaming, g= generics, mv= multiple versions, rp= requires from participant, s= state.

The “Handle creation” concern deals with the creation of handles for each figure. We placed the creation of the handles in a handle creator class that has a method for the handle creation for each class. That and the role overhead lead to more lines of code than the original implementation. But the role has an advantage over the original code: it can dynamically change the handle creator.

The “Polygon Locator” uses an anonymous class. In JavaStage roles cannot be applied to anonymous classes so we had to develop an inner class to play that role and then use it.

Unresolved Concerns. A surprising result is that for the 2 concerns with the most clone sets and class involved neither technique works. This is because they are clones in the structure and not on the code itself. The ”Creating undo activity” creates an UndoActivity object for each tool and command. Each has an UndoActivity inner class. Because inner class constructors have different parameters in number and types, roles and traits could not resolve this concern. Another example is the “Handle manipulation starting action”: code is similar but not identical: methods have different parameters. Another example is “Persistence”: because figures must be streamed they have a write and read methods with similar, but not identical, structures.

Another unresolved concern is “DrawingView listener”. The replicated code is redefining the original method for performance issues.

The unresolved “Desktop Initial configuration” deals with a Desktop’s panel initialization. Each initialization is similar so we could configure a role/trait for each. But it would be easier to know how to configure the scroll pane.

Other unresolved concern was a single line like getSomeObject().doSomething(). The first method returns different objects that call different methods, so role configuration would take more LOC.

We would count only 4 unresolved concerns if we had not considered some concerns as clones.

6.3 Threats to Validity

We only considered a single system. However, the discussion in section 5 hints that results from other systems would also have roles performing better than traits. We need to do the same test with more systems to fully assess this.

The clone detection settings can affect the clones detected which would lead to different concerns. But we needed to reduce the amount of clone sets to a manageable number or there would be a greater number of false clones. We even used less than the limit of 30 tokens recommended in [23] to limit false clones. So we believe that our settings provided a good number of clones/concerns that make this case study results valid.

There could be biased results from having the same developers doing the role and traits approach. After all the authors are more experienced in roles than in traits. This could bias the results towards roles. To prevent this, we opted to base the study on clone detection and not on developing an alternate system from scratch.

The effort used to develop each approach was not taken into account. For that we would need to assess how teams of developers, each using a different approach, tackled the same problem. While the LOC number gives a hint on the effort required,

and roles have an advantage, it does not tell the all story as already mentioned. However it does give some insights. One insight is that the use of state reduces the effort to develop roles by reducing the amount of glue code one must write to use traits. This also gives roles a better modelling capability because a role can model a concept that has state and behaviour as opposed to the traits' behaviour only modelling. We also reused roles from our role library, but traits equivalent could not be placed in a traits library, which means that the effort of developing roles was less than that of traits.

7 Related Work

There are a number of dynamic role approaches like Object Teams [14], EpsilonJ [15] and PowerJava [26]. These are known for their capability to attach and detach roles from objects at runtime, something that [17] also supports for traits. ObjectTeams introduces the notion of team. A team represents a context in which several classes collaborate to achieve a common goal. Even though roles are first class entities they are implemented as inner classes of a team and are not reusable outside that team. Roles are also limited to be played by a specific type. PowerJava has a similar concept – the institution. When an object wants to interact with an institution it must assume one of the roles the institution offers. In EpsilonJ roles are also defined as inner classes of a context. Roles are assigned to an object via a bind directive. It uses a requires directive similar to roles and traits. It also offers a replacing directive to rename methods names.

Feature Oriented Programming (FOP) [8] decomposes the system into features. FOP relies on a step-wise refinement of applications by adding new features or refining existing ones. To compose a system we just state which features it has. The composition is made automatically with tool support, like AHEAD [27]. This is a more powerful technique than roles or traits. AHEAD uses several tools for composing the code and extra files for configuring the composition step. Roles/Traits are programming languages that statically compose classes using only source code. AHEAD can be used to compose classes. For example, we can develop a class that defines the basic behaviour of a class, undistinguishable from a normal Java class, except that it has a feature keyword indicating to which feature it is associated to. We can then construct several refinements to that class. Each refinement indicates the added feature and the class it refines.

Package Templates (PT) [28] use traditional java packages with a twist. Classes defined in these packages are only directly available when the package is instantiated. When instantiated the classes can be tailored to the context of use by: getting additions; elements can be renamed; type parameters are given actual types. This tailoring is similar to roles as roles also support renaming and type parameters. PT may also impose restrictions on the various types via a constraints declaration that resembles roles requirement list. Classes in a PT can be merged with classes from other PT and can be used more than once in the same merging (like roles/traits can be used multiple times). The main difference between PT and roles is that PT, like traits, rely on inheritance to do the merging and roles rely on inner classes. Name clashes are resolved

via renaming, which can be applied to fields and methods. The renaming cannot be used on the constraints. JavaStage allows the renaming of required methods

Aspect-Oriented Programming is an approach that tries to modularize crosscutting concerns [9]. AOP defines pointcuts to identify points in the executing program that may trigger a different execution path and advices that indicate the new execution path. While the modularization of crosscutting concerns is the flagship of AOP several authors disagree [29,30]. The effects of pointcuts and advices, especially when several aspects have similar pointcuts, may be unpredictable. Thus simple changes in the class code can have unsought effects [31].

The obliviousness feature of AOP means that a class is aspect unaware so aspects can be plugged or unplugged as needed. But it introduces problems in comprehensibility [32]. To understand the system we must know the classes and which aspects affect each class. This is a major drawback when maintaining a system, since the dependencies aren't always explicit and there isn't an explicit interface between them.

With roles/traits all dependencies are explicit and the system comprehensibility is increased [12]. Roles do not have the obliviousness of AOP because the class is aware of the roles it plays.

Caesar [33] uses aspect technology to modularize crosscutting concerns and enhance reuse of aspects leading to a greater reduction of repeated code. Caesar uses an Aspect Collaboration Interface that decouples aspects binding and implementations by defining them in a separated module. Caesar does not allow method renaming.

Jiazzi [34] is based on Units [35] and aims at building systems out of reusable components integrated with the language. Jiazzi has two types of units: Atoms (composed by java classes) and Compounds (composed by atoms or other compounds). Jiazzi supports the addition of features to classes without editing their source code. Roles/Traits could be used within Jiazzi to specify these new features. A trait/role could be used to add the same behavior for different classes in the same unit, or for the same class but in different units.

8 Conclusions

We compared how the role and traits approaches deal with the composition problems that they aim to diminish by doing a study on how each can reduce replicated code, especially the replicated code that derives from lack of compositional mechanisms in single inheritance languages.

The outcome of the study showed that roles are more reusable than traits, because roles support state, have a renaming mechanism that tunes them to the class purpose and can even provide several versions of a method in a simple way. We validated our approach developing roles for the JHotDraw framework and eliminated nearly all duplicated code. Doing the same test for traits showed that they cannot eliminate all the clones roles were capable of. We even reused some roles from our role library showing that they are really reusable.

References

1. Fowler, M.: Refactoring: Improving the design of existing code. Addison-Wesley, Boston (1999)
2. Mayrand, J., Leblanc, C., Merlo, E.: Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In: Proc. of the International Conference on Software Maintenance (1996)
3. Baxter, I., Yahin, A., Moura, L., Sant'Anna, M., Bier, L.: Clone Detection Using Abstract Syntax Trees. In: Proc. of Int. Conf. on Software Maintenance (1998)
4. Roy, C., Cordy, J.: A Survey on Software Clone Detection Research. Tech. Report 2007-451, School of Computing, Queen's University at Kingston (2007)
5. Bracha, G., Cook, W.: Mixin-Based Inheritance. In: Proceedings of the OOPSLA/ECOOP, pp. 303–311. ACM Press, Ottawa (1990)
6. Ducasse, S., Schaerli, N., Nierstrasz, O., Wuyts, R., Black, A.: Traits: A mechanism for fine-grained reuse. Trans. on Programming Languages and Systems (2004)
7. Scharli, N., Ducasse, S., Nierstrasz, O., Black, A.: Traits: Composable units of behavior. In: Cardelli, L. (ed.) ECOOP 2003. LNCS, vol. 2743, pp. 248–274. Springer, Heidelberg (2003)
8. Apel, S., Kästner, C.: An Overview of Feature-Oriented Software Development. Journal of Object Technology 8(5) (July-August 2009)
9. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of aspectJ. In: Lindskov Knudsen, J. (ed.) ECOOP 2001. LNCS, vol. 2072, pp. 327–354. Springer, Heidelberg (2001)
10. Quitslund, P., Black, A.: Java with traits - improving opportunities for reuse. In: Proceedings of the 3rd International Workshop on Mechanisms for Specialization, Generalization and Inheritance (2004)
11. Smith, C., Drossopoulou, S.: *chai*: Traits for java-like languages. In: Gao, X.-X. (ed.) ECOOP 2005. LNCS, vol. 3586, pp. 453–478. Springer, Heidelberg (2005)
12. Riehle, D.: Framework Design: A Role Modeling Approach, Ph. D. Thesis, Swiss Federal Institute of technology, Zurich (2000)
13. Steimann, F.: On the representation of roles in object-oriented and conceptual modeling. Data & Knowledge Engineering 35(1), 83–106 (2000)
14. Herrmann, S.: Programming with Roles in ObjectTeams/Java. In: AAAI Fall Symposium: "Roles, An Interdisciplinary Perspective" (2005)
15. Tamai, T., Ubayashi, N., Ichiyama, R.: Objects as Actors Assuming Roles in the Environment. In: Choren, R., Garcia, A., Giese, H., Leung, H.-f., Lucena, C., Romanovsky, A. (eds.) SELMAS. LNCS, vol. 4408, pp. 185–203. Springer, Heidelberg (2007)
16. Barbosa, F., Aguiar, A.: Using Roles to Model Crosscutting Concerns. In: Aspect Oriented Software Devlopment (AOSD3), Fukuoka, Japan, March 24-29 (2013)
17. Van Cutsem, T., Bergel, A., Ducasse, S., De Meuter, W.: Adding State and Visibility Control to Traits Using Lexical Nesting. In: Drossopoulou, S. (ed.) ECOOP 2009. LNCS, vol. 5653, pp. 220–243. Springer, Heidelberg (2009)
18. Ducasse, S., Wuyts, R., Bergel, A., Nierstrasz, O.: User-changeable visibility: Resolving unanticipated name clashes in traits. In: Proceedings OOPSLA, New York, NY (2007)
19. Fanta, R., Rajlich, V.: Removing Clones from the Code. Journal of Software Maintenance: Research and Practice 11(4), 223–243 (1999)
20. Komondoor, R., Horwitz, S.S.: Semantics-Preserving Procedure Extraction. In: Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000), Boston, MA, USA, pp. 155–169 (2000)

21. Higo, Y., Kamiya, T., Kusumoto, S., Inoue, K.: Refactoring Support Based on Code Clone Analysis. In: Bomarius, F., Iida, H. (eds.) PROFES 2004. LNCS, vol. 3009, pp. 220–233. Springer, Heidelberg (2004)
22. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
23. Kamiya, T., Kusumoto, S., Inoue, K.: Ccfinder: a multilinguistic tokenbased code clone detection system for large scale source code. IEEE Trans. Soft. Eng. 28(7) (2002)
24. Ceccato, M., Marin, M., Mens, K., Moonen, L., Tonella, P., Tourwe, T.: A qualitative comparison of three aspect mining techniques. In: Proc. of the Inter. Workshop on Program Comprehension, Washington (2005)
25. Barbosa, F., Aguiar, A.: Roles as Modular Units of Composition. In: 7th International Conference on Evaluation of Novel Approaches to Software Engineering, Wroclaw, Poland, pp. 29–30 (June 2012)
26. Baldoni, M., Boella, G., van der Torre, L.: Interaction between Objects in powerJava. Journal of Object Technologies 6, 7–12 (2007)
27. Batory, D., Sarvela, J.N., Rauschmayer, A.: Scaling Step-Wise Refinement. IEEE TSE 30(6) (2004)
28. Krogdahl, S., Møller-Pedersen, B., Sørensen, F.: Exploring the use of Package Templates for flexible reuse of Collections of related Classes. Journal of Object Technology 8(7) (2005)
29. Steimann, F.: The paradoxical success of aspect-oriented programming. In: Proceedings of the 21st Annual Conference OOPSLA 2006 (2006)
30. Przybyłek, A.: Systems Evolution and Software Reuse in Object-Oriented Programming and Aspect-Oriented Programming. In: Bishop, J., Vallecillo, A. (eds.) TOOLS 2011. LNCS, vol. 6705, pp. 163–178. Springer, Heidelberg (2011)
31. Kästner, C., Apel, S., Batory, D.: A Case Study Implementing Features using AspectJ. In: 11th Inter. Conference of Software Product Line, Kyoto, Japan (2007)
32. Griswold, W.G., Sullivan, K., Song, Y., Shonle, M., Tewari, N., Cai, Y., Rajan, H.: Modular Software Design with Crosscutting Interfaces. IEEE Software 23(1), 51–60 (2006)
33. Mezini, M., Ostermann, K.: Conquering Aspects with Caesar. In: Proc. of AOSD 2003, pp. 90–99 (2003)
34. McDirmid, S., Flatt, M., Hsieh, W.C.: Jiazzi: new-Age Components for Old-Fashioned Java. In: OOPSLA 2001 (2001)
35. Flatt, M., Felleisen, M.: Units: Cool modules for HOT languages. In: Proc. of PLDI (May 1998)

Testing Distributed Communication Protocols by Formal Performance Monitoring

Xiaoping Che and Stephane Maag

Institut Mines-Telecom/Telecom SudParis, CNRS UMR 5157,
9 rue Charles Fourier, 91011 Evry Cedex, France
{xiaoping.che,stephane.maag}@telecom-sudparis.eu

Abstract. Performance testing of communicating protocols is a qualitative and quantitative test of a system, aiming at verifying whether the performance requirements of the protocol have been satisfied under certain conditions. On the other hand, conformance testing of communicating protocols is a functional test which verifies whether the behaviours of the protocol satisfy defined requirements. It raises the interesting issue of how to accurately formalize the performance requirements and how to converge these two kinds of tests by using the same formal approach. In this paper, we present a novel logic-based approach to distributively test the conformance and performance of a protocol, through real execution traces and formally specified properties. In order to evaluate and assess our methodology, we have designed a distributed testing framework and developed a prototype for testing network protocols. Finally, the relevant verdicts of experiments with a set of IMS/SIP properties and discussions are provided.

Keywords: Performance Testing, Distributed Framework, Formal Methods.

1 Introduction

In the recent years, many studies on checking the behavior of an Implementation Under Test (IUT) have been performed. Important works are about the record of the observation during run-time and its comparison with the expected behavior defined by either a formal model [1] or a set of formally specified properties [2] obtained from the requirements of the protocol. The observation is performed through Points of Observation (PO) set on monitored entities composing the System Under Test (SUT). These approaches are commonly identified as Passive Testing approaches (or monitoring). With these techniques, the protocol messages observed in execution traces are generally modeled and analyzed through their control parts [3]. In [4] and [5], a data-centric approach is proposed to test the conformance of a protocol by taking account the control parts of the messages as well as the data values carried by the message parameters contained in an extracted execution trace.

However, within the protocol testing process, conformance and performance testing are often associated. They are mainly applied to validate or verify the scalability and reliability of the system. Many benefits can be brought to the testing process if both inherit from the same approach. Our main objective is then to propose a novel passive distributed performance testing approach based on our formal conformance testing

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 110–125, 2013.

technique [5]. Although some crucial works have been done in conformance testing area [6], they study run-time verification of properties expressed either in linear-time temporal logic (LTL) or timed linear-time temporal logic (TLTL). Different from their work focusing on testing functional properties based on formal models, our work concentrates on formally testing non-functional properties without formal models. Also note that, our work is absorbed in the performance testing, not in performance evaluation. While performance evaluation of network protocols focuses on the evaluation of its performance, performance testing approaches aim at testing performance requirements that are expected in the protocol standard.

Generally, the performance testing characteristics are: volume, throughput and latency [7], where volume represents "total number of transactions being tested," throughput represents "transactions per second the application can handle" and latency represents "remote response time." In this work, we firstly extend a proposed methodology to present a passive testing approach for checking the performance requirements of communicating protocols. Furthermore, we define a formalism to specify performance and time related requirements represented as formulas tested on real protocol traces. Finally, since several protocol performance requirements need to be tested on different entities during a common time period, we design a distributed framework for testing our approach on run-time execution traces.

Our paper's primary contributions are:

- A formal approach is proposed for formally testing performance requirements of Session Initiation Protocol (SIP).
- A distributed testing framework is designed based on an IP Multimedia Subsystem (IMS) environment.
- Our approach is successfully evaluated by experiments on the Session Initiation Protocol.

The reminder of the paper is organized as follows. In Section 2, a short review of the related works are provided. In Section 3, a brief description of the syntax and semantics used to describe the tested properties is presented. In Section 4, our framework has been implemented and relevant experiments are depicted in Section 5. It has been performed through a real IMS framework to test SIP properties. The distributed architecture of the IMS allows to assess our approach efficiently. Finally, we conclude and provide interesting perspectives in Section 6.

2 Related Works

While a huge number of papers are dedicated to performance evaluation, there are very few works tackling performance testing. We however may cite the following ones.

Many studies have investigated the performance of distributed systems. A method for analyzing the functional behavior and the performance of programs in distributed systems is presented in [8]. In the paper, the authors discuss event-driven monitoring and event-based modeling. However, no evaluation of the methodology has been performed.

In [9], the authors present a distributed performance-testing framework, which aimed at simplifying and automating service performance testing. They applied Diperf to two GT3.2 job submission services, and several metrics are tested, such as *Service response time*, *Service throughput*, *Offered load*, *Service utilization* and *Service fairness*.

Besides, in [10], the authors propose an approach based on selecting performance relevant use-cases from the architecture designs, and execute them as test cases on the early available software. Finally, they conclude that the software performance testing of distributed applications has not been thoroughly investigated. An approach to performance debugging for distributed systems is presented in [11]. This approach infers the dominant causal paths through a distributed system from traces. In addition, in [12], a new distributed continuous quality assurance process is presented. It uses in-house and in-the field resources to efficiently and reliably detect performance degradation in performance-intensive systems.

In [13], the authors present a monitoring algorithm SMon, which continuously reduces network diameter in real time in a distributed manner. Through simulations and experimental measurements, SMon achieves low monitoring delay, network tree, and protocol overhead for distributed applications. Similarly, in [14], they present a performance monitoring tool for clusters of PCs which is based on the simple concept of accounting for resource usage and on the simple idea of mapping all performance related state. They identify several interesting implementation issued related to the collection of performance data on a Clusters of PCs and show how a performance monitoring tool can efficiently deal with all incurring problems. Nevertheless, these two last approaches do not provide a formalism to test a specific requirement. Our approach allows to formally specified protocol performance requirements to be tested on real distributed traces in order to check whether the tested performance is as expected by the protocol standard.

3 Formal Approach

3.1 Basics

A communication protocol message is a collection of data fields of multiple domains. Data domains are defined either as *atomic* or *compound* [5]. An *atomic* domain is defined as a set of numeric or string values. A *compound* domain is defined as follows.

Definition 1. A *compound* value v of length $n > 0$, is defined by the set of pairs $\{(l_i, v_i) \mid l_i \in L \wedge v_i \in D_i \cup \{\epsilon\}, i = 1...n\}$, where $L = \{l_1, ..., l_n\}$ is a predefined set of labels and D_i are data domains. A *compound domain* is then the set of all values with the same set of labels and domains defined as $\langle L, D_1, ..., D_k \rangle$.

Once given a network protocol P, a *compound* domain M_p can generally be defined by the set of labels and data domains derived from the message format defined in the protocol specification/requirements. A *message* of a protocol P is any element $m \in M_p$. For each $m \in M_p$, we add a real number $t_m \in \mathbb{R}^+$ which represents the time when the message m is received or sent by the monitored entity.

Example 1. A possible message for the SIP protocol, specified using the previous definition could be

$$m = \{(method, \textbf{`INVITE'}), (time, \text{`644.294133000'}),\\ (status, \epsilon), (from, \text{`alice@a.org'}), (to, \text{`bob@b.org'}),\\ (cseq, \{(num, 7), (method, \textbf{`INVITE'})\})\}$$

representing an INVITE request from *alice@a.org* to *bob@b.org*. The value of *time* '644.294133000' ($t_0 + 644.294133000$) is a relative value since the PO started its timer (initial value t_0) when capturing traces.

A *trace* is a sequence of messages of the same domain containing the interactions of a monitored entity in a network, through an interface (the PO), with one or more peers during an arbitrary period of time. The PO also provides the relative time set $T \subset \mathbb{R}^+$ for all messages m in each *trace*.

3.2 Syntax and Semantics of Our Formalism

In our previous work, a syntax based on Horn clauses is defined to express properties that are checked on extracted traces. We briefly describe it in the following. Formulas in this logic can be defined with the introduction of terms and atoms, as it follows.

Definition 2. A *term* is defined in BNF as $term ::= c \mid x \mid x.l.l...l$ where c is a constant in some domain, x is a variable, l represents a label, and $x.l.l...l$ is called a *selector variable*.

Definition 3. A *substitution* is a finite set of bindings $\theta = \{x_1/term_1, ..., x_k/term_k\}$ where each $term_i$ is a *term* and x_i is a variable such that $x_i \neq term_i$ and $x_i \neq x_j$ if $i \neq j$.

Definition 4. An *atom* is defined as

$$\begin{aligned} A ::= & \overbrace{p(term, ..., term)}^{k} \\ & \mid term = term \\ & \mid term \neq term \\ & \mid term < term \\ & \mid term + term = term \end{aligned}$$

where $p(term, ..., term)$ is a predicate of label p and arity k. The *timed atom* is a particular atom defined as $\overbrace{p(term_t, ..., term_t)}^{k}$, where $term_t \in T$.

Example 2. Let us consider the message m in Example 1. A time constraint on m can be defined as 'm.time $<$ 550'. These atoms help at defining timing aspects as mentioned in Section 3.1.

The relations between *terms* and *atoms* are stated by the definition of clauses. A *clause* is an expression of the form $A_0 \leftarrow A_1 \wedge ... \wedge A_n$ where A_0 is the head of the clause and $A_1 \wedge ... \wedge A_n$ its body, A_i being *atoms*.

A formula is defined by the following BNF:

$$\phi ::= A_1 \wedge ... \wedge A_n \mid \phi \rightarrow \phi \mid \forall_x \phi \mid \forall_{y>x} \phi \\ \mid \forall_{y<x} \phi \mid \exists_x \phi \mid \exists_{y>x} \phi \mid \exists_{y<x} \phi$$

where $A_1, ..., A_n$ are *atoms*, $n \geq 1$ and x, y are variables.

In our approach, while the variables x and y are used to formally specify the message of a trace, the quantifiers commonly define "it exists" ($\exists$) and "for all" ($\forall$). Therefore, the formula $\forall_x \phi$ means "for all messages x in the trace, ϕ holds".

The semantics used in our work is related to the traditional Apt–Van Emdem–Kowalsky semantics for logic programs [15], from which an extended version has been provided in order to deal with messages and trace temporal quantifiers. Based on the above described operators and quantifiers, we provide an interpretation of the formulas to evaluate them to $\top$ ('*Pass*'), $\bot$ ('*Fail*') or '?' ('*Inconclusive*').

We formalize the timing requirements of the IUT by using the syntax above described, and the truth values $\{\top, \bot, ?\}$ are provided to the interpretation of the obtained formulas on real protocol execution traces. We can note that most of the performance requirements are based on relative conformance requirements. For testing some of the performance requirements, both conformance and performance formulas as well as a '$\star$' operator are used to resolve eventual confusing verdicts.

Example 3. The performance requirement "*the message response time should be less than 5ms*" (can be formalized to formula ψ) is based on the conformance requirement "*The SUT receives a response message*" (can be formalized to formula φ).

Once a '$\top$' truth value is given to a performance requirement, without doubt, a '*Pass*' testing verdict should be returned for both the performance requirement and its relative conformance requirement. However, if a '$\bot$' or '?' truth value is returned for a performance requirement, we can not distinguish whether it does not satisfy the performance requirement or it does not satisfy the relative conformance requirement. For instance, in Example 3, if a '$\bot$' is given to this formalized performance requirement ψ, we can not distinguish whether it is owing to "*The message response time is greater than 5ms*" or "*The SUT did not receive a response message*". Moreover, once we have a '?' result, it is tough to resolve it by seeking the real cause. For solving these problems, we define the function $eval_\star$ providing a truth value based on the evaluation of φ and ψ.

Definition 5. Let φ and ψ be two formulas, $eval_\star$ is defined as follows:

$$eval_\star(\varphi, \psi) = \begin{cases} \top & \text{if } eval(\varphi, \theta, \rho) = \top \\ & \text{and } eval(\psi, \theta, \rho) = \top \\ ? & \text{if } eval(\varphi, \theta, \rho) = ? \\ & \text{and } eval(\psi, \theta, \rho) = ? \\ \bot & \text{otherwise} \end{cases}$$

where $eval(\varphi, \theta, \rho)$ expresses the evaluation of a formula φ, θ represents a substitution and ρ a finite trace. Due to a lack of space, we do not herein present our already published algorithm evaluating a formula φ on trace ρ. However, the interested reader may

refer to our previous publication [5]. As above mentioned, some of the performance requirements need to be tested in a distributed way. We focus on this aspect in the next section.

4 Distributed Framework of Performance Testing

4.1 Framework

For the aim of distributively testing conformance and performance requirements, we use a passive distributed testing architecture. It is defined based on the standardized active testing architectures [16] (master-slave framework) in which only the PO are implemented.

As Figure 1 depicts, it consists to one global monitor and several sub testers. In order to capture the transporting messages, the sub testers are linked to the nodes to be tested. Once the traces are captured, they will be tested through the predefined requirement formulas, and the test results will be sent back to the global monitor. On the other side, the global monitor is attached to the server to be tested, aiming at collecting the traces from the server and receiving statistic results from sub testers. The collected aggregate results will be analyzed. This should intuitively reflects the real-time conformance and performance condition of the protocol during testing procedures.

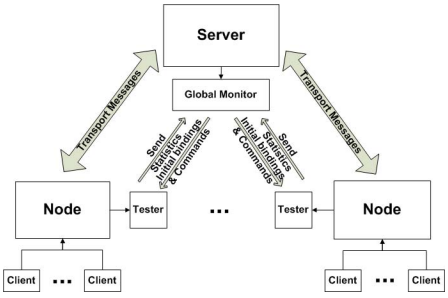


Fig. 1. Distributed testing architecture

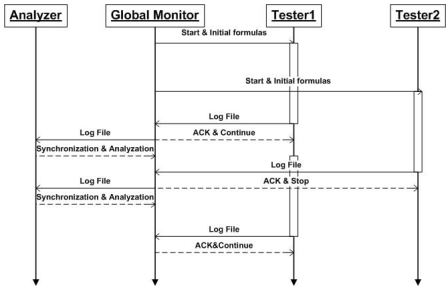


Fig. 2. Sequence Diagram between Testers

Initially, as the Figure 2 shows, the global monitor sends initial bindings (formalized requirement formulas, testing parameters) to the sub testers. When the testers receive these information, they initialize capturing packets and save the traces to readable files during each time slot. Once the readable files are generated, the testers will test the traces through the predefined requirements formulas and send the results back to the global monitor. The analyzer mentioned here is a part of the Global Monitor, for precisely describing the testing procedure, we illustrate it separately. This testing procedure will keep running until the global monitor returns the **Stop** command.

4.2 Synchronization

Several synchronization methods are provided in distributed environment [17]. Besides, Network Time Protocol (NTP) [18] is the current standard for synchronizing clocks on the Internet. Applying NTP, time is stamped on packet k by the sender i upon transmission to node j (T_{ij}^k). The receiver j stamps its local time both upon receiving a packet (R_{ij}^k), and upon re-transmitting the packet back to source (T_{ji}^k). The source i stamps its local time upon receiving the packet back (R_{ji}^k). Each packet k will eventually have four time stamps on it T_{ij}^k, R_{ij}^k, T_{ji}^k and R_{ji}^k. The computed round-trip delay for packet k is $RTT_{ij}^k = (R_{ij}^k - T_{ij}^k) + (R_{ji}^k - T_{ji}^k)$. Node i estimates its own clock offset relative to node j's clock as $(1/2)[(R_{ij}^k - T_{ij}^k) + (R_{ji}^k - T_{ji}^k)]$.

NTP is designed for synchronizing a set of entities in the networks. In our framework, relative timers are used for all the testers. However, the mismatches between these timers are ineluctable, especially the mismatches between the global monitor timer and sub tester timers would affect the results, when real-time performance is being analyzed under the influence of network events. Accordingly, the global monitor and sub testers need to be synchronized, and synchronizations between neighbor testers are not required. For satisfying the needs, slight modifications have been made to the transmission process. Rather than exchanging the four time stamps in NTP, two time duration are computed and exchanged. We choose an existing successful transaction from the captured traces, since the messages are already tagged with time stamps when captured by the monitors, the redundant tag actions can be omitted.

We use T_s to represent the service time of the server (time for reacting when receiving a message), and T_1 represents the time used for receiving a response in the client side. Benefiting from capturing traces from both Server and Client sides, the sum $(R_{ij}^k - T_{ij}^k) + (R_{ji}^k - T_{ji}^k)$ can be transformed to $(R_{ij}^k - T_{ji}^k) - (T_{ij}^k - R_{ji}^k) = T_1 - T_s$. Although relative timers are still used for each device, they are merely used for computing the time duration. After capturing the traces, two sets of messages generated: Set_{server}=[Req_i, Res_i, ... ,Req_{i+n}, Res_{i+n}] and Set_{client}=[Req_j, Res_j, ... ,Req_{j+m}, $Res_{j+m} \mid j \leq i, j+m \leq i+n$]. As we mentioned before, a successful transaction (Req_k, $Res_k \mid k \leq j+m$) will be chosen from the Set_{client} for the synchronization. The time duration T_1 of the transaction can be easily computed and sent to the global monitor with the testing results. Once the chosen transaction sequence has been found in the Set_{server}, the time duration T_s can be obtained, and the time offset $(1/2)(T_1 - T_s)$ between the global monitor and a sub tester can be handled. In the experiments, the average time used for the synchronization is about $5ms$, which provides satisfying results for our method.

4.3 Testing Algorithm

The testing algorithms are described in Figure 4.3. The algorithm on the left describes the behaviors of sub testers when receiving different commands. When the tester receives a "$Start$" command, firstly it initializes the testing parameters (line 3). Then it starts capturing the traces and tests them (as mentioned in Section 3) when traces are translated to readable xml files (lines 20-32). Finally the results are sent back to the global monitor with the chosen transaction for synchronization.

```
Require: (Tester) Received Command
Ensure: Statistic Logs
1: while Listening Port n do
2:     if Receive = Start & Initial bindings then
3:         Set Initial bindings to formulas, TimeSlot
4:         Capture(), Test()
5:         Send log(i) to Global Monitor {Send log file to
           the Global Monitor}
6:         Pending
7:     end if
8:     if Receive = Continue then
9:         Capture(), Test()
10:        Send log(i) to Global Monitor
11:        Pending
12:    end if
13:    if Receive = Stop then
14:        return
15:    else
16:        Send UnknownError to Global Monitor
17:        Pending;
18:    end if
19: end while
20: Procedure Capture(timeslot)
21: for (timer=0;timer≤time maximum;timer++) do
22:    Listening Port (5060) & Port (5061) {Capture
       packets}
23:    if timer%timeslot==0 then
24:        Buffer to Tester(i).xml {Store the packets in
           testable formats}
25:    end if
26: end for
27: Procedure Test(formulas)
28: for (j=0;j≤max;j++) do
29:    Test formula(j) through Tester(i).xml {Test the
       predefined requirement formulas}
30:    Record results to log(i) {Save the results to log
       file}
31:    Record first transaction to log(i) {Use the first
       transaction for synchronization}
32: end for
```

```
Require: (Global Monitor) Log Files
Ensure: Performance Graphs
1: Capture(), Test()
2: Display graphs
3: for (i=0;i<tester-number;i++) do
4:     Send Initial bindings to Tester[i] {Send initial bind-
       ings to all sub testers}
5: end for
6: while Listening do
7:     if command==Continue then
8:         Send Continue to Tester[i]
9:     else
10:        Send Stop to Tester[i]
11:    end if
12:    Synchronize(Log[i].transaction)
13:    Analyze(Log[i].results)
14:    Display graphs
15: end while
16: Procedure Synchronize(Log[i].transaction)
17: for (a=0; a≤Message-Number, quit!=1; a++) do
18:    find Client.Request(k) in Server.Request(a)
19:    if (exists==True) then
20:        for (b=a; b≤Message-Number, quit!=1; b++)
           do
21:            find Client. Response(k) in Server. Re-
               sponse(b)
22:            if (exists==True) then
23:                Calculate T_s
24:                Handle timer deviation (T_1 − T_s)/2
25:                quit=1
26:            else
27:                Return transaction error
28:                quit=1
29:            end if
30:        end for
31:    end if
32: end for
```

Fig. 3. Algorithm of Tester and Global Monitor

The other algorithm sketches the global monitor behaviors and the synchronization function. Initially, the monitor starts to capture and test as the other testers do. Meanwhile, it sends initial bindings to all the sub testers and waits for their responses (lines 3-6). Once the server receives the response, it reacts according to the content of the response, and the synchronization is made during this time (lines 7-15). In the $synchronize()$ procedure, the monitor finds the chosen transaction in its captured traces, and rectifies the time offset $(1/2)(T_1 - T_s)$.

5 Experiments

5.1 Environment

The IMS (IP Multimedia Subsystem) is a standardized framework for delivering IP multimedia services to users in mobility. It aims at facilitating the access to voice or

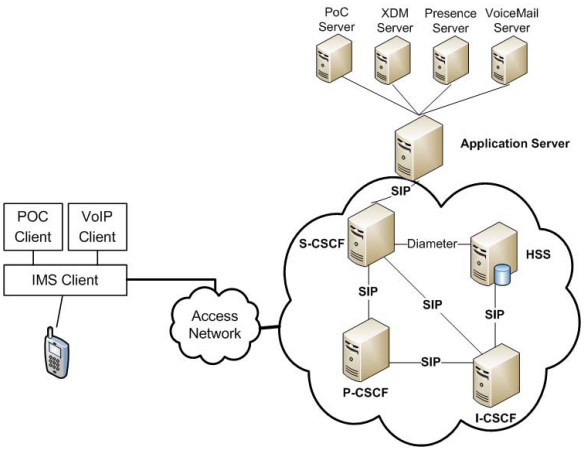


Fig. 4. Core functions of IMS framework

multimedia services in an access independent way, in order to develop the fixed-mobile convergence. The core of the IMS network consists on the Call Session Control Functions (CSCF) that redirect requests depending on the type of service, the Home Subscriber Server (HSS), a database for the provisioning of users, and the Application Server (AS) where the different services run and interoperate. Most communication with the core network and between the services is done using the Session Initiation Protocol [19]. Figure 4 shows the core functions of the IMS framework and the protocols used for communication between the different entities.

The Session Initiation Protocol (SIP) is an application-layer protocol that relies on request and response messages for communication, and it is an essential part for communication within the IMS framework. Messages contain a header which provides session, service and routing information, as well as an body part to complement or extend the header information. Several RFCs have been defined to extend the protocol to allow messaging, event publishing and notification. These extensions are used by services of the IMS such as the Presence service [20] and the Push to-talk Over Cellular (PoC) service [21].

For our experiments, traces were obtained from SIPp [22]. SIPp is an Open Source test tool and traffic generator for the SIP protocol, provided by the Hewlett-Packard company. It includes a few basic user agent scenarios and establishes and releases multiple calls with the INVITE and BYE methods. It features the dynamic display of statistics on running tests, TCP and UDP over multiple sockets or multiplexed with retransmission management and dynamically adjustable call rates. The traces obtained from SIPp contain all communications between the client and the SIP core. Tests were performed using a prototype implementation of the formal approach above mentioned, using algorithms introduced in the previous Section.

5.2 Architecture

As Figure 5 shows, a distributed architecture is performed for the experiments. It consists on one central server and several nodes. Global Monitor and sub testers are implemented to the server and nodes respectively, each node carries the traffic of numerous clients. Due to the limitation of pages, we here only illustrate the detailed results of the server and two sub testers (1&2).

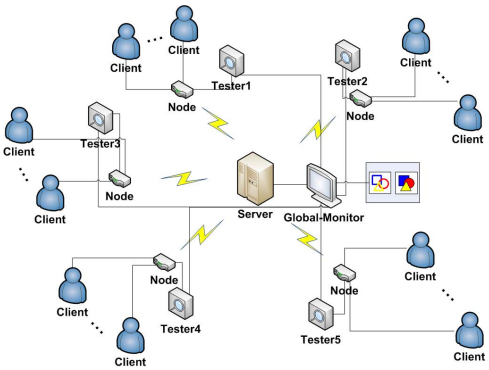


Fig. 5. Environment

5.3 Tests Results

In our approach, the conformance and performance requirement properties are formalized to formulas. These formulas will be tested through the testers. After evaluating each formula ϕ on a trace ρ, N_p, N_f and N_{in} will be given to global monitor as the results, which represent the number of '*Pass*', '*Fail*' and '*Inconclusive*' verdicts respectively. Besides, t_{slot} represents the time used for capturing a trace ρ, which is the time duration between the last and the first captured messages, where $\rho = \{m_0, ..., m_n\}$. We may write:

$$N_p(\phi) = \sum[eval(\phi, \theta, \rho) = \text{`}\top\text{'}]$$

$$N_f(\phi) = \sum[eval(\phi, \theta, \rho) = \text{`}\bot\text{'}]$$

$$N_{in}(\phi) = \sum[eval(\phi, \theta, \rho) = \text{`?'}]$$

$$t_{slot} = m_n.time - m_0.time$$

We classify the conformance and performance requirements into three sets: Session Establishment indicators, Global indicators and Session Registration indicators.

Session Establishment Indicators. In this set, properties relevant to session establishment are tested. Conformance requirements $\varphi_{a_1}, \varphi_{a_2}$ ("Every **INVITE** request must be responded", "Every successful **INVITE** request must be responded with a success response") and performance requirement ψ_{a_1} ("The Session Establishment Duration should not exceed $T_s = 1s$") are tested. They can be formalized as the following formulas:

$$\varphi_{a_1} = \begin{cases} \forall_x(request(x) \wedge x.method = \textbf{'INVITE'} \\ \rightarrow \exists_{y>x}(nonProvisional(y) \wedge responds(y,x))) \end{cases}$$

$$\varphi_{a_2} = \begin{cases} \forall_x(request(x) \wedge x.method = \textbf{'INVITE'} \\ \rightarrow \exists_{y>x}(success(y) \wedge responds(y,x))) \end{cases}$$

Table 1. Every **INVITE** request must be responded, Every successful **INVITE** request should be responded with a success response and The Session Establishment Duration should not exceed T_s

		φ_{a_1}			φ_{a_2}			ψ_{a_1}		
Tr	No.Msg	Pass	Fail	Incon	Pass	Fail	Incon	Pass	Fail	Incon
1	1164	101	0	0	85	16	0	85	16	0
2	3984	339	0	0	270	69	0	270	69	0
3	6426	520	0	0	425	95	0	425	95	0
4	7894	615	0	0	473	142	0	473	142	0
5	7651	600	0	0	477	123	0	477	123	0
6	7697	604	0	0	492	112	0	490	114	0
7	7760	607	0	0	491	166	0	490	167	0
8	7683	601	0	0	492	159	0	491	160	0
9	7544	587	2	0	464	123	0	461	126	0
10	7915	620	0	0	487	133	0	487	133	0

$$\psi_{a_1} = \begin{cases} \forall_x(request(x) \wedge x.method = \textbf{'INVITE'} \\ \rightarrow \exists_{y>x}(success(y) \wedge responds(y,x) \\ \wedge within time(y,x,T_s))) \end{cases}$$

By using these formulas, the performance indicators of session establishment are defined as:

- Session Attempt Number: $N_p(\varphi_{a_1})$
- Session Attempt Rate: $N_p(\varphi_{a_1})$ / t_{slot}
- Session Attempt Successful Rate: $N_p(\varphi_{a_1})$ / $N_p(\varphi_{a_2})$
- Session establishment Number: $N_p(\varphi_{a_2})$
- Session establishment Rate: $N_p(\varphi_{a_2})$ / t_{slot}
- Session establishment Duration: $N_p(\psi_{a_1})$.

The results of sub tester1 are illustrated in Table 1. A number of '*Fail*' verdicts can be observed when testing φ_{a_2} and ψ_{a_1}. This could indicate that during the testing time, the server refused some 'INVITE' requests and some session establishments exceeded the required time. Nonetheless, all of them can be perfectly detected by using our approach. Figure 6 illustrates the successful session establishment rates of the server and two sub

testers during the testing times. Benefited from the synchronization process, from the figure, we can observe that the curve of sub tester1 begins 1.5s later than the others. In other words, the sub tester1 started the testing process 1.5s later than the others, it might be caused by the delay of transportation or the slow response of the processor. However, it successfully shows the usage of our synchronization to precisely reflect the results of testing in distributed environment.

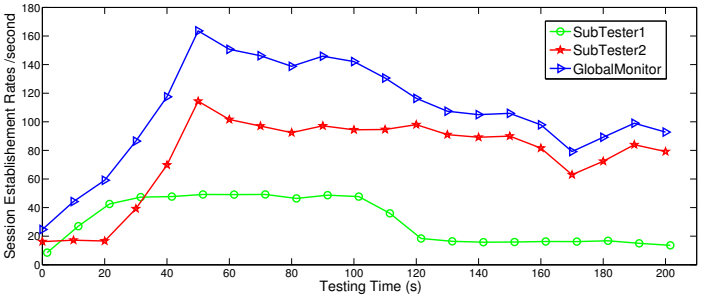


Fig. 6. Session Establishment Rates

Global Parameters. In this set, relevant properties to general network performance are tested. Conformance requirement φ_{b_1} ("Every request must be responded") and performance requirement ψ_{b_1} ("Every request must be responded within $T_1 = 0.5s$")

are used for the test, and they can be formalized as it follows.

$$\varphi_{b_1} = \begin{cases} \forall_x (request(x) \wedge x.method! = \textbf{'ACK'} \\ \rightarrow \exists_{y>x} (nonProvisional(y) \wedge responds(y, x))) \end{cases}$$

$$\psi_{b_1} = \begin{cases} \forall_x (request(x) \wedge x.method! = \textbf{'ACK'} \\ \rightarrow \exists_{y>x} (nonProvisional(y) \wedge responds(y, x) \\ \wedge within time(x, y, T_1))) \end{cases}$$

By using these formulas, several performance indicators related to general packet analysis can be formally described.

- Packet Throughput: $N_p(\varphi_{b_1})$ / t_{slot}
- Packet loss Number: $N_f(\varphi_{b_1})$
- Packet loss Rate: $N_f(\varphi_{b_1})$ / $N_p(\varphi_{b_1})$ + $N_f(\varphi_{b_1})$ + $N_{in}(\varphi_{b_1})$
- Packet Latency: $N_p(\psi_{b_1})$

The testing results of sub tester1 are shown in Table 2 and Figure 7. From Figure 7, during the time 130s to 200s, an upsurge of request rates can be observed. This one is mainly due to the burst increase of requests in sub tester2 especially since the request throughput of sub tester1 remains steady. However, compared to Figure 6, no evident increment of session establishment can be observed during the same time (130s to 200s). Indeed, during a session establishment, 'INVITE' requests represent the major part of the total number of requests. It raises a doubt about the source of the increase on these requests. With this doubt we step over to test the session registration properties.

Table 2. Every request must be responded & Every request must be responded within $T_1 = 0.5s$

		φ_{b_1}			ψ_{b_1}		
Trace	No.of msg	Pass	Fail	Incon	Pass	Fail	Incon
1	1164	258	0	0	258	0	0
2	3984	899	0	0	899	0	0
3	6426	1481	0	0	1481	0	0
4	7894	1858	0	0	1858	0	0
5	7651	1793	0	0	1791	2	0
6	7697	1802	0	0	1795	7	0
7	7760	1829	0	0	1820	9	0
8	7683	1799	0	0	1792	7	0
9	7544	1782	4	0	1766	20	0
10	7915	1855	2	0	1855	2	0

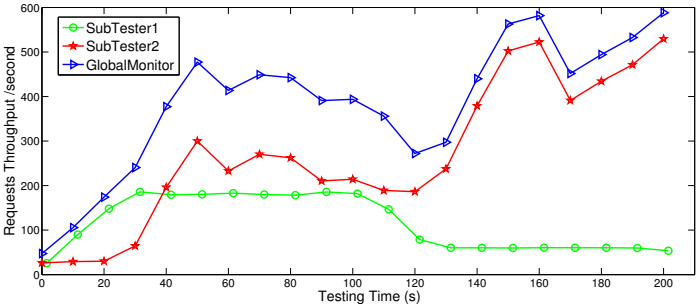


Fig. 7. Request Throughput

Session Registration. In this set, properties on session registration are tested. Conformance requirement φ_{c_1} ("Every successful **REGISTER** request should be with a success response") and performance requirement ψ_{c_1} ("The Registration Duration should not exceed $T_r = 1s$") are used for the tests.

$$\varphi_{c_1} = \begin{cases} \forall_x (request(x) \wedge x.method = \textbf{'REGISTER'} \\ \rightarrow \exists_{y>x} (success(y) \wedge responds(y, x)))) \end{cases}$$

$$\psi_{c_1} = \begin{cases} \forall_x (request(x) \wedge x.method = \textbf{'REGISTER'} \\ \rightarrow \exists_{y>x} (success(y) \wedge responds(y, x) \\ \wedge withintime(x, y, T_r))) \end{cases}$$

By using these formulas, some performance indicators related to session registration can be formally described.

- Registration Number: $N_p(\varphi_{c_1})$
- Registration Rate: $N_p(\varphi_{c_1})/t_{slot}$
- Registration Duration: $N_p(\psi_{c_1})$

The results of sub tester1 are shown in Table 3 and Figure 8. As Figure 8 depicts, there do exists an increment of registration requests during 130s to 200s. But these increased requests are not sufficient enough for eliminating the previous doubt, since deviation still exists on the number of requests. Take the peak rate at 160s for example, the server throughput nearly reaches to 600 requests/s in Figure 7, while in Figure 8 and 6, the sum of two throughput is only over 200 requests/s, even counting the 'BYE' requests, the source of the 300 other requests/s can not be defined by this analysis.

Table 3. Every successful **REGISTER** request should be with a success response & Registration Duration

		φ_{c_1}			ψ_{c_1}		
Trace	No.of Msg	Pass	Fail	Incon	Pass	Fail	Incon
1	1164	105	0	0	105	0	0
2	3984	340	0	0	340	0	0
3	6426	520	0	0	520	0	0
4	7894	614	0	0	614	0	0
5	7651	602	0	0	602	0	0
6	7697	603	0	0	599	4	0
7	7760	609	0	0	597	12	0
8	7683	602	0	0	596	6	0
9	7544	593	2	0	579	16	0
10	7915	619	2	0	619	2	0

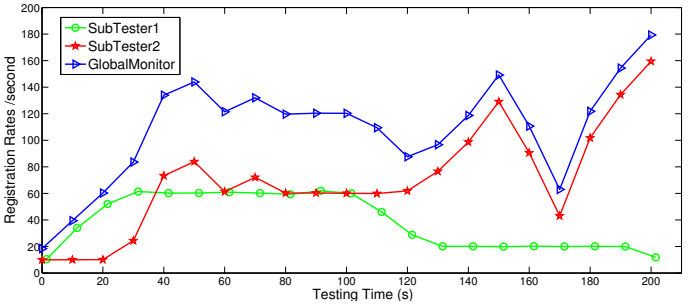


Fig. 8. Registration Rates

Nevertheless, when thinking about packet losses, our test-bed may be led to a high rate of requests with low effectiveness. In order to confirm this intuition, we check the test results of 'Request packet loss rate' property. The results are illustrated in the Figure 9. As expected, there is a high rate packet loss both in the Global monitor and sub tester2 during the time internal [130s,200s]. By taking, for instance, the same 160s sample, almost 50% of the requests are lost. It means that the actual effective throughput should be the half number of the previous test results. This finally allows to define the source of the 300 other requests/s. This also successfully shows the usage of our indicators for analyzing abnormal conditions such as burst throughput, high rate packet loss, etc.

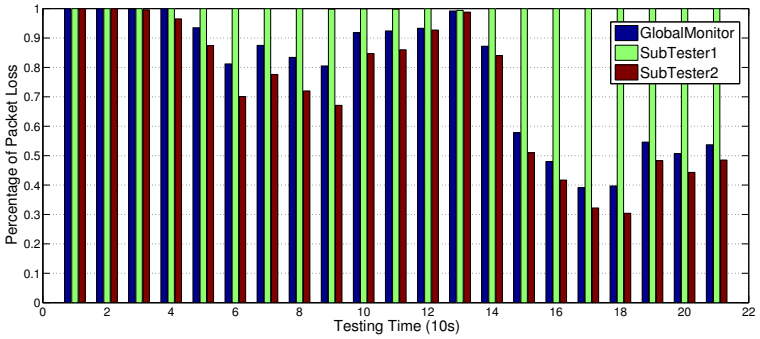


Fig. 9. Packet loss Rate

6 Perspectives and Conclusions

This paper introduces a novel approach to passive distributed conformance and performance testing of network protocol implementation. This approach allows to define relations between messages and message data, and then use such relations in order to define the conformance and performance properties that are evaluated on real protocol traces. The evaluation of the property returns a *Pass*, *Fail* or *Inconclusive* result, derived from the given trace. To verify and test the approach, we design several SIP properties to be evaluated by our approach. Our methodology has been implemented into a distributed framework which provides the possibility to test individual nodes of a complex network environment, and the results from testing several properties on large traces collected from an IMS system have been obtained with success.

Furthermore, instead of simply measuring the global throughput and latency, we extended several performance measuring indicators for SIP. These indicators are used for testing the conformance and performance of SIP in a distributed network. The real time updated results displayed in the screen can precisely reflect the performance of the protocol in different network conditions. Consequently, extending more indicators and building a standardized performance testing benchmark system for protocols would be the work we will focus on in the future. In that case, the efficiency and processing capacity of the system when massive sub testers are performed would be the crucial point to handle, leading to an adaptation of our algorithms to more complex situations.

References

1. Lee, D., Miller, R.: Network protocol system monitoring-a formal approach with passive testing. IEEE/ACM Transactions on Networking 14(2), 424–437 (2006)
2. Lalanne, F., Maag, S.: A formal data-centric approach for passive testing of communication protocols. IEEE / ACM Transactions on Networking (2012)
3. Hierons, R.M., Krause, P., Luttgen, G., Simons, A.J.H.: Using formal specifications to support testing. ACM Computing Surveys 41(2), 176 (2009)
4. Lalanne, F., Che, X., Maag, S.: Data-centric property formulation for passive testing of communication protocols. In: Proceedings of the 13th IASME/WSEAS, ACC 2011/MMACTEE 2011, pp. 176–181 (2011)

5. Che, X., Lalanne, F., Maag, S.: A logic-based passive testing approach for the validation of communicating protocols. In: Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2012, Wroclaw, Poland, pp. 53–64 (2012)
6. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for ltl and tltl. ACM Transactions on Software Engineering and Methodology 20, 14 (2011)
7. Weyuker, E.J., Vokolos, F.I.: Experience with performance testing of software systems: Issues, an approach, and case study. IEEE Trans. Software Eng. 26, 1147–1156 (2000)
8. Hofmann, R., Klar, R., Mohr, B., Quick, A., Siegle, M.: Distributed performance monitoring: Methods, tools and applications. IEEE Transactions on Parallel and Distributed Systems 5, 585–597 (1994)
9. Dumitrescu, C., Raicu, I., Ripeanu, M., Foster, I.: Diperf: An automated distributed performance testing framework. In: 5th International Workshop in Grid Computing, pp. 289–296. IEEE Computer Society (2004)
10. Denaro, G., Bicocca, U.D.M., Polini, A., Emmerich, W.: Early performance testing of distributed software applications. In: SIGSOFT Software Engineering Notes, pp. 94–103 (2004)
11. Aguilera, M.K., Mogul, J.C., Wiener, J.L., Reynolds, P., Muthitacharoen, A.: Performance debugging for distributed systems of black boxes. SIGOPS Oper. Syst. Rev. 37, 74–89 (2003)
12. Yilmaz, C., Krishna, A.S., Memon, A., Porter, A., Schmidt, D.C., Gokhale, A., Natarajan, R.: Main effects screening: a distributed continuous quality assurance process for monitoring performance degradation in evolving software systems. In: ICSE 2005: Proceedings of the 27th International Conference on Software Engineering, pp. 293–302. ACM Press (2005)
13. Yuen, C.H., Chan, S.H.: Scalable real-time monitoring for distributed applications. IEEE Transactions on Parallel and Distributed Systems 23, 2330–2337 (2012)
14. Taufer, M., Stricker, T.: A performance monitor based on virtual global time for clusters of pcs. In: Proceedings of IEEE International Conference on Cluster Computing, pp. 64–72 (2003)
15. Emden, M.V., Kowalski, R.: The semantics of predicate logic as a programming language. Journal of the ACM 23(4), 733–742 (1976)
16. 9646-1, I.: ISO/IEC information technology - open systems interconnection - conformance testing methodology and framework - part 1: General concepts. Technical report, ISO (1994)
17. Shin, M., Park, M., Oh, D., Kim, B., Lee, J.: Clock synchronization for one-way delay measurement: A survey. In: Kim, T.H., Adeli, H., Robles, R., Balitanas, M. (eds.) ACN 2011. CCIS, vol. 199, pp. 1–10. Springer, Heidelberg (2011)
18. Mills, D.L.: Internet time synchronization: the network time protocol. IEEE Transactions on Communications 39, 1482–1493 (1991)
19. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J.: Sip: Session initiation protocol (2002)
20. Alliance, O.M.: Internet messaging and presence service features and functions (2005)
21. Alliance, O.M.: Push to talk over cellular requirements (2006)
22. Hewlett-Packard: SIPp (2004), `http://sipp.sourceforge.net/`

Research in Global Software Engineering: A Systematic Snapshot

Bilal Raza, Stephen G. MacDonell, and Tony Clear

SERL, School of Computing & Mathematical Sciences, Auckland University of Technology, Private Bag 92006, Auckland 1142, New Zealand
{bilal.raza,stephen.macdonell,tony.clear}@aut.ac.nz

Abstract. This paper reports our extended analysis of the recent literature addressing global software engineering (GSE), using a new Systematic Snapshot Mapping (SSM) technique. The primary purpose of this work is to understand what issues are being addressed and how research is being carried out in GSE – and comparatively, what work is not being conducted. We carried out the analysis in two stages. In the first stage we analyzed 275 papers published between January 2011 and June 2012, and in the second stage we augmented our analysis by considering a further 26 papers (from the *2013 International Conference on Global Software Engineering* (ICGSE'13). Our results reveal that, currently, GSE studies are focused on management- and infrastructure-related factors, using principally evaluative research approaches. Most of the studies are conducted at the organizational level, mainly using methods such as interviews, surveys, field studies and case studies. The USA, India and China are major players in GSE, with USA-India collaborations being the most frequently studied, followed by USA-China. While a considerable number of GSE-related studies have been published since January 2011 they are currently quite narrowly focused, on exploratory research and explanatory theories, and the critical research paradigm has been untouched. An absence of formulative research, experimentation and simulation, and a related focus on evaluative approaches, all suggest that existing tools, methods and approaches from related fields are being tested in the GSE context, even though these may not be inherently applicable to the additional scale and complexity of GSE.

Keywords: Global Software Engineering (GSE), Distributed Software Development, Classification, Systematic Mapping.

1 Introduction

Interest in software development carried out by globally distributed, culturally and/or temporally diverse teams arose with the advent of outsourcing in the last two decades, and it continues to increase [1]. Its importance has led to the emergence of the specific area of research and practice referred to as global software engineering (GSE) [1]. GSE is itself a growing field as is clearly evident in the diversity of locations involved and the rapidly increasing number of published studies into GSE-related issues. As the number of such studies increases it becomes important to periodically summarize

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 126–140, 2013.

the work and provide overviews of the results [2] as a means of reflection on what work is being done and what gaps might exist.

In this paper we investigate the breadth of topics that have been covered by GSE studies over a short timeframe, using a variant of the systematic mapping (SM) method that we refer to as a *systematic snapshot*. This approach establishes a specific baseline state that could be further extended in a backward or forward direction to analyse changes over time. The systematic mapping (SM) method has been widely used in medical research [2] and was first adopted in software engineering research by Bailey et al. [3]. A SM aims to provide a high-level view of the relevant research literature by classifying the work according to a series of defined categories and visualizing the status of a particular field of research [2] [4]. This technique has been used recently in the GSE field [4] [5] [6] [7] [8]. In these studies specific aspects of GSE research were categorized (using guidelines presented in [2] and [9]). These investigations considered between 24 and 91 primary studies, published up to the year 2010. The aspects of GSE analyzed in these studies were software configuration management, awareness support, agile practices, project management, and tools in GSE. All five studies therefore classified the GSE literature from a relatively narrow perspective but covering a wide temporal range. They were published in well-known journals and conferences and provide valuable contributions to the body of GSE literature. In our study, we instead use a new variant of the systematic mapping process called Systematic Snapshot Mapping (SSM), briefly described in the next section, to classify the very recent global software engineering literature.

The rest of this paper is organized as follows: in Section 2 we describe our research approach in greater detail, and in Section 3 we present the findings of our analysis. In the subsequent Section 4 we briefly discuss validity threats. In Section 5 we conclude this paper and Section 6 conveys future work.

2 Method and Conduct

The results presented in this paper derive from our classification of the current literature on GSE, using the Systematic Snapshot Mapping (SSM) method. In order to classify this literature we chose the time period between January 2011 and June 2012 and later extended it to include papers published in the Proceedings of ICGSE'13. This study followed guidelines presented by Petersen et al. [2] for carrying out systematic mapping studies. However, instead of narrowing down the topic and considering a large temporal period, we limited the time span and considered the full breadth of topics covered. This study was inspired by several prior classifications of SE and GSE literature including that of Glass et al. [10], but instead of following a random sampling technique to select papers (as in [10]) we used a systematic process. We employed a defined protocol for choosing search strings and executing them against relevant databases to cover the breadth of GSE-related studies. We defined our categories at the outset of our work and chose various dimensions to present the results, mainly leveraging the prior classifications of Richardson et al. [11] and Glass et al. [10]. We present our results in the form of tables, bar graphs and network analysis graphs to provide visual representations of the data. We believe such a snapshot approach is especially useful in cases where a field is changing rapidly and where there

is consequently rapid growth in the research literature. This new approach for carrying out systematic mapping also provides an opportunity to effectively build upon different researchers' work by using different temporal ranges.

2.1 Research Questions

The following research questions were established for this study:

- RQ1. What are the factors, levels and locations investigated in the recent GSE literature?
- RQ2. How is research being carried out in GSE in regard to methods and approaches?

2.2 Search Strategy

Our search strategy was designed to keep the topic general while addressing a short time period to provide an up-to-date overview of the research literature. Initial search keywords were selected from known GSE systematic literature reviews and mapping studies. These keywords were updated based upon various dry runs carried out on the Scopus database to ensure their effectiveness. In the initial run, a target was set to ensure at least those studies from which the keywords were taken were retrieved. In the second run, a random set of ten studies was selected from the Proceedings of the 2009-2011 ICGSE conferences, and the search strings were further refined to ensure that these sample studies were also retrieved.

Table 1 shows the final list of keywords used to cover as many variations of the same term as possible. We intentionally adopted many keywords having low precision but high recall [12] and subsequently complemented our analysis by including all the papers published in ICGSE'13.

Table 1. List of keywords used as search strings

"global software engineering" OR "global software development" OR "distributed software engineering" OR "distributed software development" OR "offshore software development" OR "offshore software engineering" OR "distributed team" OR "global team" "offshore insourcing" OR "geographically distributed teams" OR "global software" OR {("software" OR "information system" OR "computer" OR "information technology") AND ("virtual team" OR "dispersed team" OR "far shore" OR "offsite" OR "offshore outsource" OR "outsourced"}|

2.3 Data Sources and Retrieval

We searched across multiple data sources to retrieve as many potentially relevant studies as possible. SCOPUS, IEEE Xplore, the ACM Digital Library, SpringerLink and ScienceDirect were searched to complement results. Each database has limitations in terms of the number of keywords accepted at a specific instance; therefore, we had to break the search phrases to suit the particular database. The initial search and retrieval process was conducted in July 2012 and the date range was limited to

January 2011 to June 2012. The search was carried out on metadata (title, abstract, keywords) and only peer-reviewed literature published in English was considered. In the first step, citations of retrieved studies were downloaded and duplicates were removed. Afterwards, the studies were then considered for the inclusion process.

2.4 Inclusion Process

The steps taken in the inclusion process to select primary studies are shown in Figure 1. After searching all of the databases 2020 studies were retrieved. The decision for further inclusion was based upon the first author's reading of the papers' titles or abstracts (resulting in 1125 studies). Duplicates were then removed, and a full text version of each remaining study was sought. For 12% of the papers (53 of the 437 remaining) the full text was not available to us, primarily because the papers were not published in well-known journals or conference proceedings. These studies were therefore not considered for further analysis. The full text of the remaining 384 papers was then reviewed by the first author and a set of 275 studies was selected for inclusion in the SM analysis. Studies in the form of short papers, extended abstracts and position papers (only describing future work) were excluded. A number of studies, not related to the software engineering domain, had slipped through to this stage and upon cursory review of the full text were also excluded. At this stage, we also considered the papers of ICGSE'13 and included them in our final list for analysis.

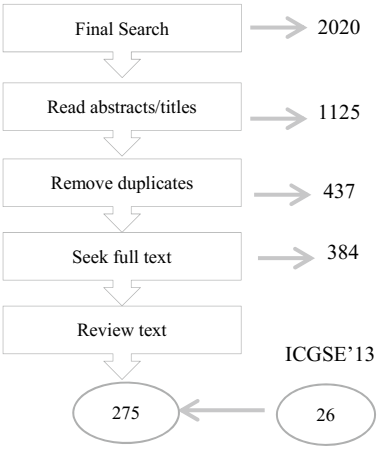


Fig. 1. Study inclusion process

2.5 Data Extraction and Synthesis

We followed generally accepted guidelines [2] to build our SM classification scheme. The included studies were therefore categorized according to various dimensions: research approach, research method, factors, level of analysis, sourcing phases and locations. In order to reduce threats to validity, regular meetings of the three authors were held to discuss issues and address misconceptions. In order to reduce bias effects

the three researchers also conducted a sample classification together. At a later point a further sample of studies which were initially classified by the first author were verified by the senior researchers, discussions were held again and issues were addressed. It was established that the authors were in general agreement regarding the classification, based upon the sample results.

The classification scheme utilized by Glass et al. [10] was used to characterize the research approach for our set of studies. We also considered the same source for the methodologies used in software engineering research. However, to better reflect the GSE perspective we also considered other methodologies [1][13]. Hence, we added Computer Mediated Communication (CMC) analysis to cover studies that investigate artifacts such as chat-histories and emails. Although grouped together in prior studies, Observations and Interviews were considered separately, as many studies use them to complement other methods. Interviews are widely used as a sub-method in Case Studies and Observations are used in Ethnographies. However, we observed that these methods are being used in their own right and we therefore classified them separately. We included the method Data Analysis to signify studies that utilize data from Repositories, Incident Management Systems and Archives of previous projects. We used Proof of Concept for non-empirical studies in which entities were formulated but were only described by examples rather than any formal validation.

3 Findings

This section presents the results obtained based on the classifications of the data extracted from our final combined set of studies.

3.1 Findings for Factors

Richardson et al. [11] identified 25 GSE factors in an empirical study and grouped them in the four broad categories of Distance, Infrastructure, Management and Human Factors. We used these categories to also characterize our identified studies. We added Learning/Training/Teaching, Competition and Performance to the Management category and Relationship to the Human Factors category. We also updated the latter category with Coordination/collaboration. Table 2 presents the results of this classification. The results clearly show that recent GSE studies are heavily focused on Management- and Infrastructure-related factors compared to Human- and Distance-related factors. Šmite et al. [1] presented a systematic review of empirical GSE research and also found that most of the studies were focused on management-related issues. Comparing these results with the SWEBOK [14] knowledge areas (KAs), it was found that the standard lacks specific considerations for GSE. As a corollary, it was also found that KAs related to design, construction, testing and maintenance are not widely addressed in the recent GSE literature.

Table 2. Findings for GSE factors and their percentage

Distance	***16.4%***	Team Selection	0.8%
Communication	8.4%	Effective Partitioning	4.6%
Language	1.1%	Skills Management	0.4%
Culture	5.3%	Knowledge transfer/knowledge management	6.1%
Temporal issues	1.6%	Visibility	3.3%
Human Factors	***16.03%***	Reporting Requirement	0.0%
Fear	0.4%	Information Management	1.1%
Motivation	2.1%	Teamness	5.5%
Trust	2.7%	Learning/Training/teaching	4.2%
Cooperation	1.6%	Competition	0.6%
Coordination/collaboration	7.8%	Performance	1.8%
Relationship	1.2%	***Infrastructure***	***24.2%***
Management	***43.2%***	Process Management	8.2%
True Cost	1.7%	Tools	9.1%
Project Management	8.8%	Technical Support	0.4%
Risk Management	2.3%	Communication tools	6.5%
Roles and responsibilities	1.6%		

3.2 Findings for Research Approach

GSE presents a complex context that demands a more extensive repertoire of research methods and approaches than those currently prevailing [15]. Table 3 presents the findings of our classification of the research approaches used in current GSE-related studies. In terms of the three main categories, the dominant research approach is Evaluative, followed by Descriptive and then Formulative. This is in sharp contrast to the results reported in 2002 by Glass et al. [10] in which the order was Formulative, Descriptive and Evaluative. One of the main reasons for the present dominance of Evaluative research is the inclusion of new empirical methods such as CMC analysis, Interviews, Data Analysis and Observations. These results appear to be in contrast with the results of Šmite et al.'s systematic review [1] of GSE-related studies published between 2000 and 2008. They concluded that GSE-related studies are relatively small in number and immature and most of them focused on problem-oriented reports. Our current results show, however, that GSE publications have grown in quantity and quality and more studies have used evaluative approaches. Of note is that these evaluative approaches are mostly confined to previously formulated work. We interpret this to mean that existing methods, tools and so on from related fields, such as collocated software engineering (CSE), are being evaluated in the context of GSE. Given that GSE is fundamentally different from CSE[11], it seems likely that solutions formulated for CSE will need to be updated or enhanced for GSE. Entirely new solutions may also need to be identified and assessed in the GSE context.

Similarly, there is clear potential for critical research in this context particularly in light of the power structures that can exist between GSE 'partners', and the associated issues of trust, fear, cooperation and the like (as shown in Table 2). Criteria or principles for carrying out critical research are lacking generally in information systems (IS) [16]. Considering its importance, Myers and Klein [16] proposed a set of principles for conducting critical research – these principles could be considered in future investigations of human factors in GSE.

Table 3. Findings for research approach

Research Approach	Percentage	Research Approach	Percentage
Descriptive	***25.4%***	Evaluative-other	12.1%
Descriptive-system	7.4%	***Formulative***	***18.5%***
Review of literature	9.8%	Formulative-framework	5.2%
Descriptive-other	8.1%	Formulative-guidelines/standards/approach (FG)	1.6%
Evaluative	***56.1%***	Formulative-model	5.9%
Evaluative-deductive	17.6%	Formulative-process, method, algorithm	2.3%
Evaluative-interpretive	26.1%	Formulative-classification/taxonomy	0.5%
Evaluative-critical	0.2%	Formulative-concept	2.7%

3.3 Findings for Research Methods

Figure 2 depicts the research methods used. The most dominant methods are Interview, Survey, Field Study and Case Study, indicating that most of the studies employed qualitative methods. These results are also in stark contrast to more general SE classifications [10] in which researchers used very few case or field studies. For studies in which multiple methods were used we assigned more than one research approach and method. Research methods in GSE are currently skewed towards exploratory research focusing on theories relating to 'Explanation' as described by Gregor [17]. These theories aim to provide explanation about what, how and why things happen and to promote greater understanding of phenomena. Thus, although GSE research has grown in terms of the number of studies being conducted, these studies are exploratory and/or explanatory in nature. It will be interesting to compare these results with future studies to determine whether work moves towards more predictive studies as the field matures.

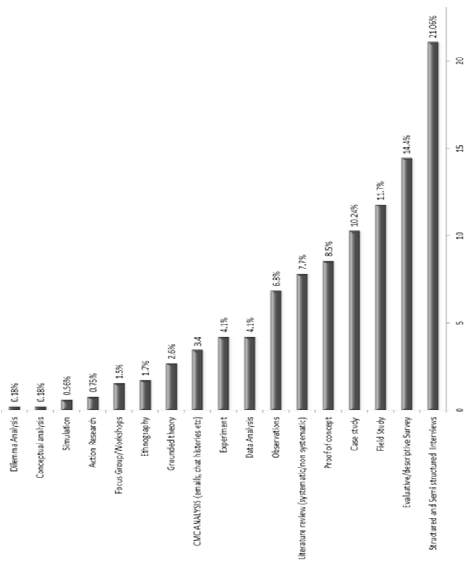


Fig. 2. Findings for research methods

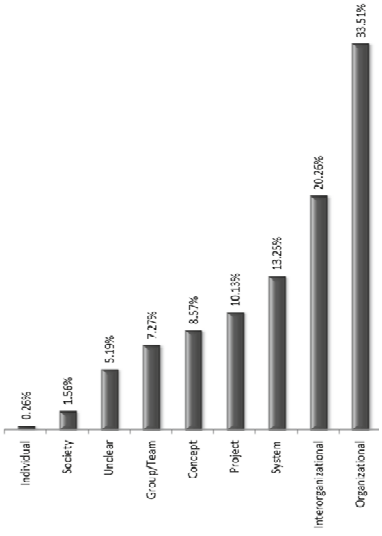


Fig. 3. Findings for level of analysis

Table 4. Distribution of studies across Journals, Conferences and Workshops

Journals		IEEE TEM	2	ISEC	3
		LNBIP	2	ICSSP	3
IST Journal	8	J Grp Dec Negot	2	MySEC	2
JSEP	7	**Conferences**		EUROMICRO	2
J Softw. Maint. Evo.	7			ICIS	2
IET Software	6	ICGSE	52	CollaborateCom	2
J of E Markets	4	HICSS	15	CTS	2
IEEE Software	4	ICSE	8	PACIS	2
J Comm and Com Sc.	3	CSCW	8	**Workshops**	
ISJ	3	PROFES	6	CTGDSD	13
IJoPM	2	CHI	5	ICGSE	13
JSW	2	XP	4	CHASE	7
POM Journal	2	ICIC	3	OTM	3
IS	2	PICMET	3	Global Sourcing	3

3.4 Findings for Level of Analysis and Distribution of Studies

Figure 3 shows the level of analysis considered currently by GSE researchers. The dominant level of analysis was found to be Organizational followed by Inter-Organizational - combined together they are used in more than half the studies reviewed. Fewer studies addressed group, individual and societal levels, a finding that coincides with the results of Glass et al. [10] in respect of SE studies. Table 4 presents the distribution of studies across various conferences, journals and workshops with frequency greater than one. (This limit was imposed due to space considerations and for ease of interpretation.) The majority of the selected studies was published in conference proceedings and drew on an industrial context.

3.5 Bubble Plot Analysis

The use of visual techniques in SM, such as bubble plots, has been recommended by Petersen et al. [2] and such techniques have been used to convey the results of mapping and classification studies[13][6]. Figure 4 presents the results of this study in the form of a bubble plot. We chose to represent three classification dimensions within it: Research approach is on the right X-axis, GSE-factors, grouped in their four major categories, are on the Y-axis, and level of analysis is on the left X-axis. The results clearly show that most of the recent studies are focused on using evaluative approaches around management and infrastructure factors and analyzed at the organizational levels. Studies based upon specific groups, societies and individuals are limited. Organizational concerns have been at the forefront in terms of the level of analysis, leaving much scope for consideration of groups and individuals in future studies.

3.6 Location of GSE Projects and Inter-country Relationships

Figure 6 and Table 5 provide graphical and tabular representations of the locations involved in GSE projects. A few studies also mentioned regions rather than countries;

we also considered them in our analysis. Figure 5 shows the results of our examination of inter-country networks. We used NodeXL, an extendable tool kit used for data analysis and visualizations [18]. Table 6 lists the pairwise relationships with frequency greater than one. (This constraint was imposed due to space limitations; however, all the relationships are shown in Figure 5.) It can be seen in Figure 5 and Table 6 that the most connected nodes are the USA and India. Some studies explicitly mentioned the collaborating locations whereas others only specified the locations involved without clearly stating which actively collaborated. For the latter studies, we assumed pairwise relationships between each location. For future studies we recommend that authors clearly state the nature of each party's involvement.

In Figure 6, countries and regions marked by darker shades are those most frequently involved in GSE. For ease of analysis we grouped these countries into six categories based upon the number of studies that cite their involvement. Not unexpectedly, the two countries reported as most frequently involved in global software projects are the USA and India. Countries including Germany, Finland, China, the UK, Australia and Brazil are ranked in the second group, closely followed by a group comprising Sweden, the Netherlands, Japan, Argentina, Spain, Canada and Switzerland. In the next two categories lie the potentially upcoming and emerging countries of Russia, Eastern European countries such as Lithuania, Far Eastern countries including Malaysia and Indonesia, and the South/Central American countries of Chile and Mexico. These representations give some insight into the diversity of countries' involvement in GSE projects. Some of these regions are underrepresented but this does not necessarily mean that these locations are not involved in GSE; it could be that these regions have simply not been considered in recent studies.

Researchers rely on personal contacts in their national industries to validate their results. Our study also shows that the top seven locations of GSE authors are the USA, Finland, Germany, Spain, Brazil, India and Sweden. Apart from Spain, which is thirteenth, all six other countries are in the list of top ten locations involved in GSE projects. We also analyzed the inter-country collaboration of GSE researchers from different countries and found that researchers from European countries have mostly collaborated with other European-based researchers whereas researchers from the US have collaborated with European and Asian researchers.

Table 5. Locations involved in GSE projects

Country	#	Country	#	Country	#	Country	#	Country	#
US	246	Esp	18	Mys	9	Pan	5	Grc	3
Ind	167	Can	16	Mex	9	Aut	4	Twn	3
Deu	61	Che	16	Sen	9	Est	4	Rom	2
Fin	56	Ukr	15	SGP	9	Phl	4	Svk	2
Chn	44	Rus	13	Nzl	9	Tha	4	Tur	2
UK	41	Dnk	13	Hun	8	Vnm	4	Pak	2
Aus	34	Irl	12	Khm	7	Kor	4	Bgd	1
Bra	32	Ita	11	Fra	7	Pol	4	Zaf	1
Swe	27	Nor	11	Bel	7	Cri	3	Tun	1
Nld	23	Cze	10	Chl	6	Col	3		
Jpn	21	Ltu	10	Hrv	6	Ecu	3		
Arg	19	Isr	9	UAE	5	Egy	3		

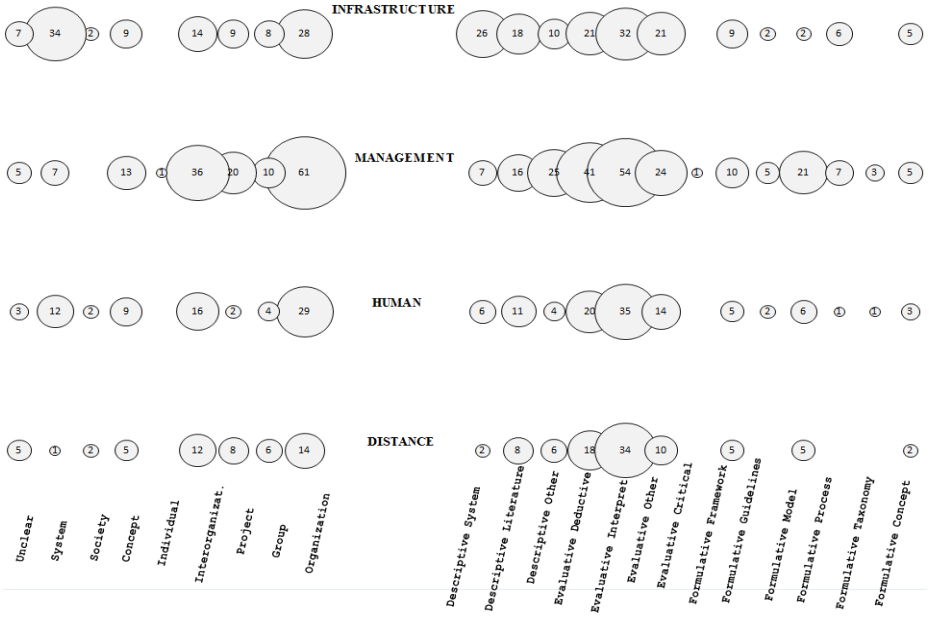


Fig. 4. Bubble plot analysis

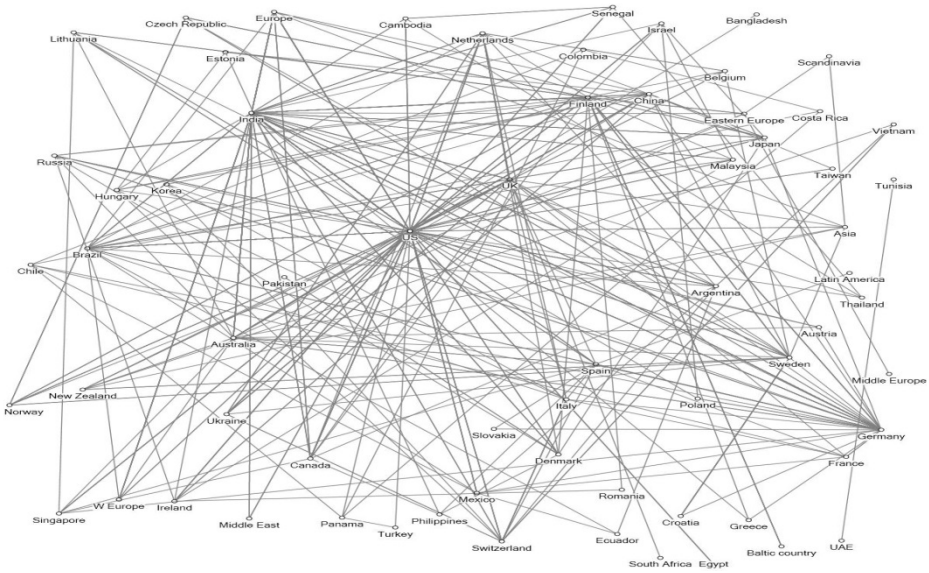


Fig. 5. Inter-country relationship analysis

Table 6. Inter-country relationships

Loc_A	Loc_B	#	Loc_A	Loc_B	#	Loc_A	Loc_B	#	Loc_A	Loc_B	#
Ind	US	69	W.Eu	Ind	6	Ita	Che	3	Eu	Jpn	2
Chn	US	23	Brazil	Ind	4	Jpn	Ind	3	Fin	Jpn	2
Deu	Ind	15	Fin	Deu	4	Nor	Fin	3	Fin	Bra	2
Bra	US	11	Ind	Swe	4	Esp	Deu	3	Fra	Deu	2
Aus	US	10	Ind	Arg	4	US	Che	3	Deu	Cze	2
Eu	US	10	Nld	US	4	US	Swe	3	Ind	Che	2
Uk	US	10	Nld	Ind	4	US	Esp	3	Ind	M.East	2
Deu	US	9	SGP	US	4	US	Sen	3	Irl	Chn	2
Uk	Ind	8	US	Ukr	4	Chn	Jpn	3	Ltu	US	2
			US	SGP	4	US	Mex	3	Mys	Ind	2
Fin	Ind	8	US	Rus	4	US	Mys	3	Nld	Ukr	2
Aus	Ind	7	US	Isr	4	US	Egy	3	Asia	US	2
Ind	Eu	7	US	Nor	4	US	Dnk	3	Nzl	US	2
Fin	US	7	Ind	Jpn	4	Nld	UK	3	Nor	Swe	2
US	Arg	7	E.Eu	Fin	3	Aus	Esp	2	Nor	Cze	2
US	Ukr	6	Fin	Swe	3	Aus	Deu	2	Esp	Ltu	2
US	Can	6	Fin	Ltu	3	Bel	US	2	Che	Vnm	2
Hrv	Swe	5	Fin	Baltic.	3	Bra	UK	2	Che	Ukr	2
Cze	Fin	5	Deu	Rus	3	Khm	Senl	2	US	Twn	2
Jpn	US	5	Deu	Bra	3	Khm	Ind	2	US	M.East	2
Swe	Hrv	5	Ind	Sen	3	W Eu	US	2	US	Khm	2
US	Jpn	5	Dnk	Ind	3	Can	Ind	2			
US	Irl	5	Ind	Chn	3	Can	Eu	2			

3.7 Phases in Sourcing Relationships

Dibbern et al. [19] divided the sourcing process into two main stages: the decision stage, which is concerned with the 'What', 'Why' and 'Which' questions, and the implementation stage addressing 'How' and 'Outcome'. This covers the processes of deciding on and managing the sourcing resulting agreement, but leaves out the transition process. Butler et al. [20] divided this same process into three main phases, of Decision, Transition and Operation, based upon the timeline of a project. Butler et al. then [20] categorised 116 articles based upon the focus of attention of GSE projects and found that only 2 articles from the 116 were related to the transitional phase. This coincides with the results of this systematic snapshot mapping study in which we categorised 301 articles across various dimensions and found that only 19 were related to transition, further highliting that limited research has been directed towards this phase.

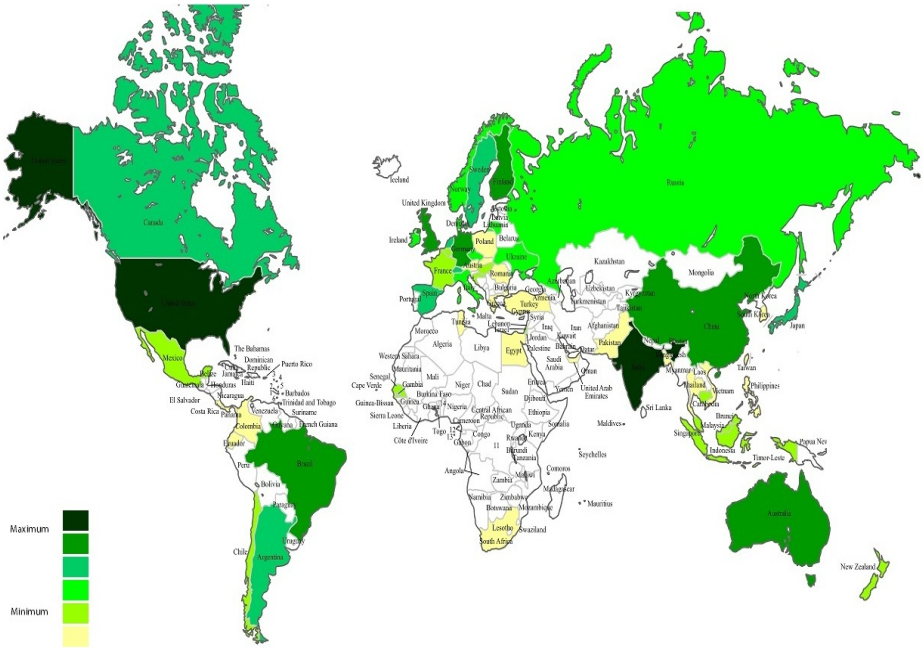


Fig. 6. Locations involved in GSE projects

4 Threats to Validity

One of the main threats to the validity of our study is the incomplete selection of primary studies or missing relevant studies, even though we followed a systematic process. In order to mitigate this risk we formulated a wide variety of search-terms. These terms were taken from related systematic mapping and systematic literature review (SLR) studies and were updated based upon the retrieved results. Initially, we ensured that at least those SM/SLR studies were indeed retrieved using the search terms drawn from each study. In the next stage, we constructed a sample list of studies from various ICGSE proceedings and ensured that the search terms retrieved these studies as well. During this process the search terms were continuously updated until all sample studies were retrieved, similar to the approach taken by [6]. A second validity threat arises due to researcher bias during the classification process. In order to reduce this threat, we carried out some sample classifications collectively. Furthermore, the lists of studies as classified by the first author were validated by the senior researchers involved. A high level of agreement was achieved, giving us confidence that the classification process was executed appropriately and consistently.

5 Conclusions

Through this study we have provided a current snapshot of the recent GSE-related research literature. We first classified 275 empirical and non-empirical studies, published between January 2011 and June 2012, into predefined categories (see

http://tinyurl.com/GSE-Papers), and we then augmented our analysis with the consideration of the papers published in ICGSE'13. We examined the following characteristics: GSE factors, research approaches, research methods, level of analysis, and GSE project locations. The GSE factors most frequently researched were related to management and infrastructure using evaluative approaches and taking an organizational perspective as the level of analysis. Regarding research methods, interviews, surveys, case studies and field studies are the most commonly used. In relation to project locations, the USA and India are the predominant nations involved in global software projects. Inter-country network analysis also shows that USA-India collaboration is at the top followed by that between the USA and China. It will be interesting to carry out further similar snapshot studies on an on-going basis to see if or how these trends evolve. Similarly, studies could be carried out retrospectively on previous years' research literature to enable comparisons with this study. This study aims to provide a stepping stone for such related studies.

It appears that, in general, existing solutions are being applied in a GSE context, even though these solutions may lack specific considerations needed for GSE. For instance, aspects of non-functional requirements and stage/phase-related issues are not addressed separately in the current GSE literature. Although the field of GSE research has grown rapidly in terms of the number of studies conducted, these studies are quite narrowly focused towards exploratory research and the provision of explanatory theories. Furthermore, in spite of GSE providing a natural and potentially fruitful setting for critical research, such work is yet to be conducted. The current research focus is mainly directed to organizational concerns, leaving much scope for consideration of the needs of stakeholder groups and individuals. The research is also skewed towards projects having two locations, showing a dearth of studies relating to multiple locations and their underlying complex relationships. Finally, there are regions of the world that are not being currently studied by researchers and it may be useful to consider them in the future studies, particularly if the dimensions of culture and their impact on GSE are of interest.

6 Future Work

A notable omission in the current focus of work relating to GSE is any sustained coverage of issues to do with power and exploitation. While the human factors tabulated in Table 2 above include some focus on the factors of fear, trust, cooperation and relationship, these are given relatively limited attention. Again in Figure 4 there is a noted absence of studies at an individual unit of analysis. There are no studies giving personal narratives or biographies – are the workers in GSE deliberately kept invisible? Is this absence a function of the research methods used, for instance, no examples of critical evaluative work have been identified in this review? Or is it an abrogation of our duties as academics to act in the role of 'critic and conscience of society'? Will the future see more equal partnerships in sustainable global ventures, or will there be a backlash against crude models of global labor arbitrage? What risks might that pose to a multi-billion dollar industry? These issues warrant more attention by researchers, although difficult to confront. In addition such research will be challenging to design and conduct, yet the absence of critical evaluative studies presents a glaring gap in current GSE research.

References

1. Šmite, D., Wohlin, C., Gorschek, T., Feldt, R.: Empirical evidence in global software engineering: a systematic review. Empirical Software Engineering 15, 91–118 (2009)
2. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic Mapping Studies in Software Engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, pp. 68–77 (2008)
3. Bailey, J., Budgen, D., Turner, M., Kitchenham, B., Brereton, P., Linkman, S.: Evidence relating to Object-Oriented software design: A survey. In: First International Symposium on Empirical Software Engineering and Measurement, pp. 482–484 (2007)
4. Fauzi, S.S.M., Bannerman, P.L., Staples, M.: Software Configuration Management in Global Software Development: A Systematic Map. In: Proceedings of the 17th Asia Pacific Software Engineering Conference, pp. 404–413 (2010)
5. Steinmacher, I., Chaves, A.P., Gerosa, M.A.: Awareness Support in Distributed Software Development: A Systematic Review and Mapping of the Literature. Computer Supported Cooperative Work 22, 113–158 (2013)
6. Jalali, S., Wohlin, C.: Agile Practices in Global Software Engineering - A Systematic Map. In: Proceedings of the 5th International Conference on Global Software Engineering, pp. 45–54 (2010)
7. Silva, F.Q.B., Prikladnicki, R., França, A.C.C., Monteiro, C.V.F., Costa, C., Rocha, R.: An evidence-based model of distributed software development project management: results from a systematic mapping study. Journal of Software: Evolution and Process 24, 625–642 (2012)
8. Portillo-Rodríguez, J., Vizcaíno, A., Piattini, M., Beecham, S.: Tools used in Global Software Engineering: A systematic mapping review. Information and Software Technology 54, 663–685 (2012)
9. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering. EBSE Technical Report EBSE-2007-01 (2007)
10. Glass, R.L., Vessey, I., Ramesh, V.: Research in software engineering: an analysis of the literature. Information and Software Technology 44, 491–506 (2002)
11. Richardson, I., Casey, V., McCaffery, F., Burton, J., Beecham, S.: A Process Framework for Global Software Engineering Teams. Information and Software Technology 54, 1175–1191 (2012)
12. Dieste, O., Padua, A.G.: Developing Search Strategies for Detecting Relevant Experiments for Systematic Reviews. In: First International Symposium on Empirical Software Engineering and Measurement, pp. 215–224 (2007)
13. Smite, D., Wohlin, C., Feldt, R., Gorschek, T.: Reporting Empirical Research in Global Software Engineering: A Classification Scheme. In: Proceedings of the Third International Conference on Global Software Engineering, pp. 173–181 (2008)
14. Guide to the Software Engineering Body of Knowledge. IEEE Computer Society (2004)
15. Clear, T., MacDonell, S.G.: Understanding technology use in global virtual teams: Research methodologies and methods. Information and Software Technology 53, 994–1011 (2011)
16. Myers, M.D., Klein, H.K.: A Set of Principles For Conducting Critical Research In Information Systems. MIS Quarterly 35, 17–36 (2011)

17. Gregor, S.: The nature of theory in information systems. MIS Quarterly 30, 611–642 (2006)
18. Smith, M.A., Shneiderman, B., Milic-Frayling, N., Mendes Rodrigues, E., Barash, V., Dunne, C., Capone, T., Perer, A., Gleave, E.: Analyzing (social media) networks with NodeXL. In: Proceedings of the Fourth International Conference on Communities and Technologies, pp. 255–263 (2009)
19. Dibbern, J., Goles, T., Hirschheim, R., Jayatilaka, B.: Information Systems Outsourcing: A Survey and Analysis of the Literature. ACM SIGMIS Database 35, 6–102 (2004)
20. Butler, N., Slack, F., Walton, J.: IS/IT Backsourcing - A Case of Outsourcing in Reverse? In: Proceedings of the 44th Hawaii International Conference on System Sciences, pp. 1–10 (2011)

Test City Metaphor for Low Level Tests Restructuration in Test Database

Artur Sosnówka

Faculty of Computer Science, West Pomeranian University of Technology,
ul.Żołnierska 49, Szczecin, Poland
arsosnowka@wi.zut.edu.pl

Abstract. The process of validating modified software to detect whether new deviations have been introduced to previously tested code is known as regression testing. Since the expense for this process conducted at the system integration level are very high very few researches has proposed formal test selection criteria at this level. Formal test selection criteria for system or integration test based on visualization analysis for low level test cases has been included in this paper. Presented analysis criteria shows a subset of test metrics which has been used in pilot projects in the industry as a base for testware reorganization.

Keywords: Visualization Metaphor, Testware, Test City, Test Metrics, Test Management, Data Mining, Test Case Visualization, Low Level Test Case, Test Selection.

1 Introduction

Software development is dealing with growing complexity, shorter delivery times and current progress made in the hardware technology. Within the software lifecycle the biggest, however not directly seen, part is the maintenance. Increasing number of systems used in the corporation and tolerated number of deviations is decreasing when time progressing and users get trusted to the used software. As soon as software is put in the production environment, every big change or even small adaption of the source code can cause potential danger in best case, monetary, in worst image or even human being loses. Nevertheless the maintenance is very often provided during the whole period through different groups of technicians or business partners. This makes the task of programming, understanding and maintaining of the source code for the system and its testware more complex and difficult.

To be successfully introduced each software system requires properly defined requirements. Those can and are very often changing during the whole project or software lifecycle. The changes are based on legal, business, functional or software architectural needs (e.g. new programming techniques). Required new functionality is gaining focus and the old one is put aside and threatened to not be as important as before. Testware management, especially for the high (HLTC) and low level test cases (LLTC) [8], which are focusing on old but still valid functionality keeps going to be not affordable, or getting be forgotten by purpose. The situation is causing

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 141–150, 2013.

raised maintenance costs to the limit, when new development can produce less cost and even be easier to implement than creation of the new functionality within the old system.

Required quality of the software is very often to be reached through quality assurance activities on several levels, starting from unit test, through system, integration and ending on acceptance tests. Artefacts produced during the test process required to plan, design, and execute tests, such as documentation, scripts, inputs, expected outcomes, set-up and clear-up procedures, files, databases, environment, and any additional software or utilities used in testing are named, according to ISTQB, testware (ISTQB, ISTQB® Glossary of Testing Terms, 2012). Detection of the problems within a testware can save much effort and reduce necessary maintenance costs. Number of executed tests in the first or second year of software maintenance is not being a disruptive factor for the test projects. As soon as software is coming into the last phase, associated teams are very often moved to the other development projects or taken out of the company (e.g. consultants are being moved from customer to customer). To prove necessary quality after performed adaptations, growing complexity of the system is demanding high professional skills and understanding from people and organizations taken over the responsibility for the system.

Software quality is according to IEEE definition:

1. The degree to which a system, component or process meets specified requirements.
2. The degree to which a system, component or process meets customer or user needs or expectations [2].

Above given definition is obligating quality assurance teams to perform planned and systematic pattern of actions to provide adequate confidence to the product or item that it conforms to established technical requirements [2]. Execution of needed actions to provide at least same quality during the whole maintenance phase is a big cost factor. According to survey-analysis presented during the iqnite 2011 conference in Düsseldorf, almost 60% of the software projects are spending between 20 and 30% of its budget on Quality Management (QM) and testing activities. Right handling of created artefacts is not a question of an effort but a need for efficiency and effectiveness.

Especially big and complex systems are providing large number of functions and demanding even larger number of objects within the testware. To provide 100% fulfilment the test team has to ensure that each function is not affected through the code adaptation and its site effects. Adaptation of the system demands adaptation of testware to fulfil quality requirement for the current system.

Even best managed testware, after few years of usage, is not free of objects which are old, obsolete, duplicated or there are no HLTCs or LLTCs covering demanded functionality. Those objects are causing additional management effort and its existence does not increase expected quality needs.

Often developers and managers believe that a required change is minor and attempt to accomplish it as a quick fix. Insufficient planning, design, impact analysis and testing may lead to increased costs in the future. Over time successive quick fixes may degrade or obscure the original design, making modifications more difficult [7] and finishing in not acceptable, low quality of the system.

As long as we are accepting loose of the software and testware quality, its transparency, increasing maintenance costs, decreasing test efficiency, and continuous testware erosion is not a subject. However, in time of financial crisis and decreasing IT budgets, there is none of the project which can come over this dilemma. In the next chapters we would like to show results from pilot project which has been executed in the industry in order to prove usefulness for the approach of the visualization metaphor for testware reorganization.

2 Related Work

Since the early days of software visualization, software has been visualized at various levels of detail, from the module granularity seen in Rigi [14] to the individual lines of code depicted in SeeSoft [3].

The increase in computing power over the last 2 decades enabled the use of 3D metric-based visualizations, which provides the means to explore more realistic metaphors for software representation. One such approach is poly cylinders [12], which makes use of the third dimension to map more metrics. As opposed to this approach in which the representations of the software artefacts can be manipulated (i.e., moved around), our test cities imply a clear sense of locality which helps in viewer orientation. Moreover, our approach provides an overview of the hierarchical (i.e., package, test object) structure of the systems.

The value of a city metaphor for information visualization is proven by papers which proposed the idea, even without having an implementation. [16] Proposed this idea for visualizing information for network monitoring and later [15] proposed a similar idea for software production. Among the researchers who actually implemented the city metaphor, [9], [1], [19] represented classes are districts and the methods are buildings. Apart from the loss of package information (i.e., the big picture), this approach does not scale to the magnitude of today's software systems, because of its granularity.

The 3D visual approach closest in focus to ours is [10], which uses boxes to depict classes and maps software metrics on their height, colour and twist. The classes' box representations are laid out using either a modified tree map layout or a sunburst layout, which split the space according to the package structure of the system. The authors address the detection of design principles violations or anti-patterns by visually correlating outlying properties of the representations, e.g., a twisted and tall box represents a class for which the two mapped metrics have an extremely high value. Besides false positives and negatives, the drawbacks of this approach is that one needs different sets of metrics for each design anomaly and the number of metrics needed for the detection oftentimes exceeds the mapping limit of the representation (i.e., 3). The detection strategies [13] were introduced as a mechanism to formulate complex rules using the composition of metrics-based filters, and extended later [11] by formalizing the detection strategies and providing aid in recovering from detected problems.

3 Visualization Metaphor

A visualization metaphor is defined as a map establishing the correspondence be tween concepts and objects of the application under test and a system of some similarities and analogies. This map generates a set of views and a set of methods for communication with visual objects in our case - test cases [6].

Lev Manovich has said: “an important innovation of computers is that they can transform any media into another”. This gives us possibility to create a new world of data art that the viewer will find as interesting. It does not matter if the detail is important to the author; the translation of raw data into visual form gives a viewer possibility to get information which is the most important just for him. Hence, any type of visualization has specific connotations, which may become metaphoric when seen in context of a specific data source. Metaphor in visualization works at the level of structure, it compares the composition of a dataset to a particular conceptual construct, and the choice of any visualization is always a matter of interpretation.

Numerous currently existing visualization systems are divided into three main classes:

- Scientific visualization systems [4];
- Information visualization systems [5]
- Software visualization systems [17].

Although all visualization systems differ in purposes and implementation details, they do have something common; they manipulate some visual model of the abstract data and are translating this into a concrete graphical representation.

In this paper we are not aiming to present all possible visualization metaphors, as this is not the focus for our research. We would like to show basic and easy to understand “City metaphor” which is helpful for representation specific test data and allow easier test reorganization. After some of the previous research work which is however not in focus of this paper we settled our first attempt to the metaphor which is very widely presented in [20] and is a part of his Phd [18]. In its research and implementation for software source code classes are represented as buildings located in city districts which in turn represent packages, because of the following reasons:

- A city, with its downtown area and its suburbs is a familiar notion with a clear concept of orientation.
- A city, especially a large one, is still an intrinsically, complex construct and can only be incrementally explored, in the same way that the understanding of a complex system increases step by step. Using an all too simple visual metaphor (such as a large cube or sphere) does not do justice to the complexity of a software system, and leads to incorrect oversimplifications: Software is complex; there is no way around this.
- Classes are the cornerstone of the object-oriented paradigm, and together with the packages they reside in, the primary orientation point for developers.

Fig. 1. Example of "Software City" representation for Android system

In our attempt we perform mapping between available LLTC and its basic metrics to provide easy to understand and manage overview about the current state of testware.

3.1 Test Metrics

To be able to perform data visualization, defined set of the static and dynamic data has to be prepared. Based on the available information's for LLTC we can extract following basic metrics, which would be used later for mapping:

- Amount of LLTC
- Execution status for available LLTC
- Last modification date/age
- Number of executions
- Number of steps
- Description length
- Execution cost
- Complexity
- Risk
- Priority

Dependent on the metrics type, those are to be taken as a data export through the available API from the test management tool or statistical data taken from the support or test organization.

Fetched metric can be mapped into the chosen visualization metaphor as:

- Data physical properties (colour, geometry, height mapping, abstract shapes)
- Data granularity (unit cubes, building border or urban block related)
- Effect of Z axis mappings on the image of the city
- Abstraction of data and LOD are key issues
- Resulting "data compatible" urban models are much larger than the original VR urban models.

4 Test Reorganization and Test Mining

In this paper we would like to show how useful can be usage of visualization based on the “Test City” metaphor. We would like to show how to perform test reorganization based on the very basic set of metrics available in the test project.

For our experimental work we have established a new system interacting with several Test Management applications placed on the market. The base idea of the system is an automation extraction and pre-evaluation of several different test metrics. Those metric are imported via available API connections from the Test Management tool and evaluated to get required set of metrics. The test metrics are provided as a text file, e.g. CSV (Comma Separated Values), and imported into visualization framework. Visualization framework allows us performing necessary analysis. The analysis result is taken as an input to the Test Management tool for Test-Set creation and evaluation.

Within our research for three test projects that contains over 4000 LLTC each, we have performed analysis for basic and extended test metrics. Those projects have been running independently with large number of common requirements. This allows us to gain information’s which are valuable to prove our concept and create inputs for further work on possible visualization usage in test management domain.

Visualization results for one of those test projects with testware structure shown in the tables 1 and 2 are shown in the Figure 2 and 4. Parameters have been based on following test metrics:

1. Test execution age → mapped to the colour.
2. Number of executions → mapped to the height.
3. Number of steps → mapped to size.

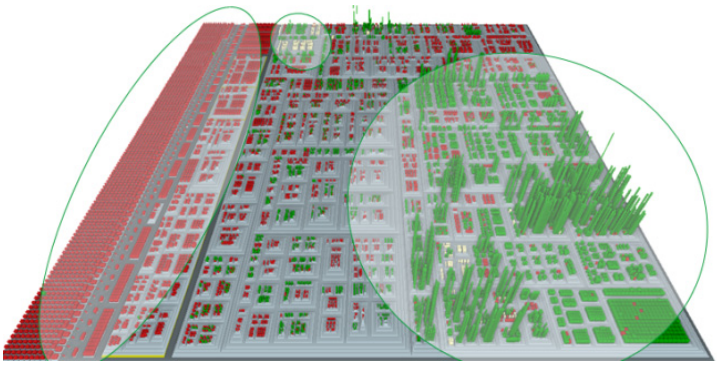


Fig. 2. Test City based on LLTC for Test Project

To provide real reference to the analysed testware, the districts (as a square group) of the Test City are mapped to the structure created by test teams and managed with help of the Test Management system (e.g. Test folder or Test object).

Looking at the possible analysis for testware visualization according to the Figure 3 we can provide following input for the improvements:

1. There is a large number of old LLTC which has been executed later than threshold set to 3000 days (red buildings – left circle in the Figure 2). Most of them had a small height which gives as an information about low number of executions. Those LLTC shall be either archived, or completely removed from the testware. LLTC not executed for longer than 9 years and rarely executed is with very high probability obsolete.
2. In the middle top, there is a circle pointing to some amount of LLTCs which has to be taken under closer investigation (yellow buildings). Execution or those objects has been done in the range of 400 to 3000 days in the past. Based on the height we can assume, most of them are obsolete; however moving to the archive is better option than leaving them within the testware.
3. Circle on the right side of the Figure 2 shows us area which has been most likely commonly used in the last 400 days. Large number of high and green buildings allows us to assume area of regression tests. Those LLTC has been used in the last period to assure certain quality of the product and shall not be moved to the archive or adapted within the first phase for testware reorganization.

Below, the tables show the visualized artefacts in numbers.

Table 1. Testware quantity for given Test project

Object type	Quantity
LLTC	18473
Executions	38128

Table 2. Testware – quantity structure

Number of executions	LLTC	(%)
0	11519	62,36
1 ... 10	5995	32,45
10 … 30	584	3,16
31 … 1000	439	2,38

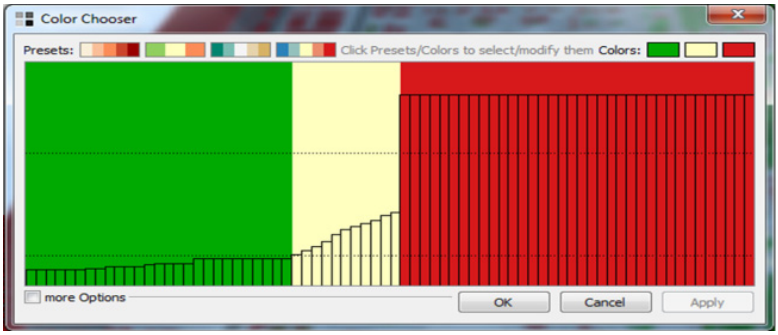


Fig. 3. Testware characteristics, looking at LLTC execution age

Figure 3 shows testware characteristics for LLTC last executions as follows:

- green → 1..380 days (~30%)
- yellow → 400… 3300 days (~15%)
- red → 3300 days (~55%)

Using a visualization to show up hotspot without possibility to localize exact coordinates cannot be used in further reorganization process. In order to localize objects within the testware we are focusing the interesting area with help of built in zoom function. Please see Figure 4 for an example.

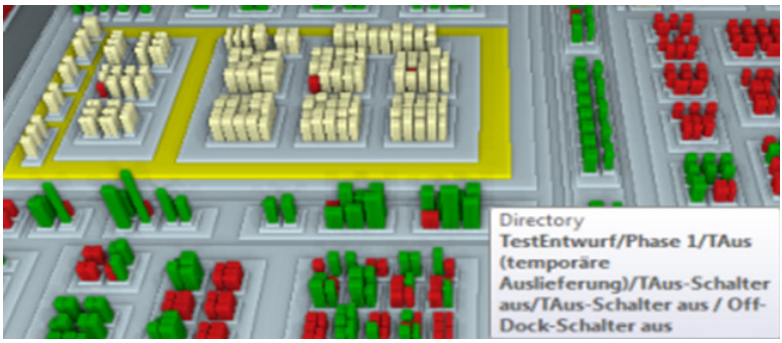


Fig. 4. Zoom for LLTCs executed between 400 and 3000 days in the past

Without having a deep knowledge about the current testware and objects details we can provide the test managers with exact information regarding that LLTCs. Currently used metrics are very basic but are giving very good start for testware reorganization and have been taken as a feedback for involved test managers.

5 Feedback from Test Managers

Created results have been presented to the involved Test managers and their feedback has been checked. Following results has been achieved:

- There is no false positives, all ugly layouts represents real problems
- No false negatives, no beauty layout should be ugly
- Unique global overview on the testware landscape
- Identify of hotspots (“there was always a question”)
- Identify cluster of issues (e.g. regression test)
- Identify cluster of stagnation

The feedback has proven our first impression we got by looking at the testware visual representation. Even if the system looks well-organized, in spite of the numerous disharmonious artefacts: we see a districts, where the test which were executed more than 365 days ago are localized and districts of increased number of high building, even skyscrapers, in which several very important and common tests are defined.

The skyscrapers are giving us the impression how many of existing LLTC have been executed very often. Their colour shows execution age as an important factor for testware reorganization.

Within very short time we were able to locate and show large number of obsolete and suspicious LLTCs. Identified hotspots and pain points based on very basic test metrics has been confirmed by the personal working for longer time with the testware, even without our deeper knowledge for the system itself. Necessary data for LLTC adaptation and/or reorganization has been exported based on zooming information at interesting areas/districts given to the test managers and used for next iteration.

6 Conclusions

Test case management, test analysis and test creation are the most important tasks within the whole test management process. It is very hard to concentrate the analysis on small set of the LLTC as it is not getting potential win against the requirement spectrum. Possible loss of testware quality can be threated only as additional cost factor and each activity steering against is helping to keep those on needed level. Performed visualization has shown us, how easy in use and efficient can be presented method for testware analysis. Finding an obsolete LLTC based on available metrics is very comfortable and does not require deep system knowledge, even if analysed system seems to be very complex. Getting the fast overview about large number of LLTCs without deep knowledge of testware saves needed time, resources and allows problem presentation not only on technical but as well on management level. Presented results have been used for further deeper analysis and reorganization activities.

Additionally we have observed person performing analysis is tending to point its view on maximum two metrics in time and not searching for further information on the third one. This behaviour was partly driven via visualization framework and its available mapping attributes and partly human laziness.

Our future directions will focus on the points listed below:

1. Extension for more APIs to Test Management tools available on the market.
2. Comparison for analysis outcome when using same metrics but different Visualization Metaphors.
3. Visualization for metrics within the timeline.
4. Extend number of evaluated metrics, especially to find out duplicate tests.

References

1. Charters, S.M., Knight, C., Thomas, N., Munro, S.: Visualisation for informed decision making; from code to components. In: Proceedings of SEKE 2002, pp. 765–772. ACM Press (2002)
2. Dickinson, W.: The Application of Cluster Filtering to operational testing of Software. Doctoral dissertation. Case Western Reserve University (2001)
3. Eick, S., Graves, T., Karr, A., Marron, J., Mockus, S.: Does code decay? Assessing the evidence from change management data. IEEE Transactions on Software Engineering 27(1), 1–12 (1998)

4. Friendly, M.: Milestones in the history of thematic cartography, statistical graphics, and data visualization (2008), http://www.math.yorku.ca/SCS/Gallery/milestone/milestone.pdf
5. González, V., Kobsa, A.: Benefits of Information Visualization Systems for Administrative Data Analysts. In: Proceedings. Seventh International Conference on Information Visualization, IV 2003, pp. 331-336, (2003).
6. Huffaker, B., Hyun, Z., Luckie, M.: IPv4 and IPv6 AS Core: Visualizing IPv4 and IPv6 Internet Topology at a Macroscopic Scale in (2010), http://www.caida.org/research/topology/as_core_network/
7. IEEE, 1059-1993 - IEEE Guide for Software Verification and Validation Plans, http://standards.ieee.org/findstds/standard/1059-1993.htm
8. ISTQB, ISTQB® Glossary of Testing Terms (2012), http://www.istqb.org/downloads/finish/20/101.html
9. Knight, C., Munro, M.C.S.: Virtual but visible software. In: 2000 IEEE Conference on Information Visualization, pp. 198–205. IEEE CS Press (2000)
10. Langelier, G., Sahraoui, H.A., Poulin, P.S.: Visualization-based analysis of quality for large-scale software systems. In: Proceedings of ASE 2005, pp. 214–223. ACM Press (2005)
11. Lanza, M., Marinescu, R.S.: Object-Oriented Metrics in Practice. Springer (2006)
12. Marcus, A., Feng, L., Maletic, J.I.: 3d representations for software visualization. In: Proceedings of SoftVis 2003, pp. 27–36. ACM Press (2003)
13. Marinescu, R.S.: Detection strategies: Metrics-based rules for detecting design flaws. In: Proceedings of ICSM 2004, pp. 350–359. IEEE CS Press (2004)
14. Muller, H., Klashinsky, S.: Rigi: A system for programming-in-the-large. In: Proceedings of ICSE 1988, pp. 80–86. ACM Press (1988)
15. Panas, T., Berrigan, R., Grundy, J.S.: A 3d metaphor for software production visualization. In: International Conference on Computer Visualization and Graphics Applications, IV 2003, vol. 314. IEEE CS Press (2003)
16. Santos, C.R.D., Gros, P., Abel, P., Loisel, D., Trichaud, N., Paris, J.P.S.: Mapping information onto 3d virtual worlds. In: Proceedings of the IV International Conference on Information Visualization 2000, pp. 379–386 (2000)
17. Stasko, J.T., Patterson, C.: Understanding and characterizing software visualization systems. In: Proceedings 1992 IEEE Workshop, pp. 3–10 (1992)
18. Wettel, R.: Software Systems as Cities. In: Doctoral Dissertation, Faculty of Informatics of the Università della Svizzera Italiana (2010)
19. Wettel, R., Lanza, M.: Visually Localizing Design Problems with Disharmony Maps. In: Proceedings of the 4th ACM Symposium on Software visualization, SoftVis 2008. ACM Press (2008)
20. Wettel, R., Lanza, M.: Visualizing Software Systems as Cities. In: Proceedings of VISSOFT 2007 (4th IEEE International Workshop on Visualizing Software For Understanding and Analysis), pp. 92–99. IEEE Computer Society Press (2007)

Service Retrieval for Service-Oriented Business Process Modeling

Youcef Baghdadi[1] and Ricardo Pérez-Castillo[2]

[1] Department of Computer Science, Sultan Qaboos University,
PO Box 36 – PC 123, Al-Khod, Oman
ybaghdadi@squ.edu.om
[2] Alarcos Research Group, University of Castilla-La Mancha,
Paseo de la Universidad, 4 13071, Ciudad Real, Spain
Ricardo.PdelCastillo@uclm.es

Abstract. Many enterprises are not able to adapt to changing business requirements. One of the solutions to this agility problem is the usage of service-oriented BP modeling. Meanwhile, their existing BP modeling does not consider the potential services in Legacy IS (LIS) or from partners, in order to have a service-oriented BP modeling that promotes agility. This requires a complete reengineering of the LIS and the BPs into services realized by business objects. In this modeling paradigm, BPs are represented by specialized services, having separated concerns such as controller service, state service, and worker services. This paper provides guidance, by using techniques to retrieve business knowledge embedded in LIS and transform it into services towards moving from as-is to to-be BPs. These techniques are: (i) reverse engineering LIS, by extracting services from traces of BPs, and (ii) reverse engineering from the enterprise service portfolio or reusing partner and provider services.

Keywords: Service-orientation, BP Modeling, Legacy Information Systems, Reverse Engineering; Service Retrieval.

1 Introduction

Many enterprises are not able to adapt quickly to uncertain, changing business requirements which is harmful in the enterprises' competitiveness. These non-agile enterprises slowly adapt to changing business requirements, which could be harmful and affects their competitiveness. Their Business Processes (BPs) are not flexible and could not adapt to the changes. Indeed, the agility is not possible if all the instances of a same BP run the same way or the BPs are not flexible enough.

To become agile, enterprises have to adapt their BPs in response to different Business Events (BEs) such as different customer demand, new acquisition, or merging. The achievement of this agility is difficult owing to: (i) BP modeling considers that all the instances of the same BP run the same way, (ii) the siloed architecture of the existing IS supporting the BPs, and (iii) the non-interoperability of the elements of the IS.

One of the solutions is to reengineer the BPs moving them towards a service-oriented BPs by using a service-oriented business modeling combining the roles of Service Orientation (SO), SOA and Web services in promoting flexibility and agility.

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 151–163, 2013.

First, we define a BP as a basic or composite service provided as a value to any of the stakeholders. A composite service uses a state that reflects its execution. This service is itself realized by enterprise IS that will play the role of service provider. Then, we represent the BP by using specialized services. These are: (i) Controller Service (CS), (ii) State Service (SS), and (iii) Worker Services (WKs), where the CS uses the SS to get the state that determines which WK to invoke [1]. In this modeling, the services are themselves provided by shared, discoverable, reusable enterprise assets known as Business Objects (BOs) that would play the role of service units [2]. We extend the OMG definition, where a BO is "a representation of a thing active in the business domain, including at least its business name and definition, attributes, behavior, relationships, and constraints" by adding a new type of BO, we refer to as Business Artifact (BA) that has a name and a state. Thus, BOs provide BPs with activities, data, and state as services. Finally we envision to represent the IS by a set of two types of BOs.

This service-oriented BP perspective requires an integrated view of all the reengineering phases [3] i.e., reverse engineering, restructuring, and forward engineering. The reverse engineering first transforms the existing IS into services, or extracts services from running BPs (if any traces). Next, during the restructuring stage, the services are categorized into one of the three types: CS, SS, or WK; and assigned to their respective BO. Then new business requirements are met to achieve agility. Finally, the forward engineering phase is based on MDA to take profit of the rapid transformation of the BP model into executable standards such as web services, BPEL, and WS-CDL.

This work limits to a reverse engineering that (i) transforms the LIS into BOs, including the existing enterprise service portfolio into some of the specialized services, (ii) extract services from existing BPs (if any), and (iii) reusing partner and provider services, or (iv) all of them.

- The reverse engineering would modernize the IS so that it provides the required worker services.
- The mapping would map the existing service portfolio into specialized services, including the controller and worker services.
- The extraction would abstract the existing BPs.

This approach would have practical implications in terms of (i) reuse of the existing assets of an organization to rapidly modeling, designing, enacting, and executing BPs that realize values, and (i) compliance with SOA high maturity level for full integration, composition, and flexibility of BPs that respond to changing event, which promotes agility.

The remainder of this paper is organized as follows: Section 2 provides some related works. Section 3 details the concepts of the BP modeling. Section 4 develops the reverse engineering approach. Finally, Section 5 discusses further development.

2 Related Work

Managing BPs as services has recently started making its way in IS by promoting BPs as compositions of loosely coupled services having separated concerns. This work concerns with two perspectives: (1) modeling BPs as services, (2) modernizing

supporting ISs with respect to SOA. From service-oriented BP modeling, the academics have raised the importance of service-oriented as one of the top three issues to deal within BPs [4]. In [5], the author proposed a business model for B2B integration through Web services. The authors in [6] proposed an approach for designing BPs in service-oriented way, where a service composition process organizes the discovery, selection and assembly of business services to build BPs tailored to business designer's requirements. In [7], the author investigated how to extend Event-Process Chain (EPC) to come up with a new modeling language for service-oriented BP management.

From a general perspective of modernizing ISs and BPs, different approaches have been proposed [8]. In [9], the authors developed Marble to reengineer BPs within a BP archeology. This approach makes it possible to obtain BP models from source code and other software artifacts by reverse engineering. Efforts that concern with modernization for SOA have been reported in [10]. In [11], the author proposed to reverse engineer relational databases into services. In [12-13] the authors have defined a technique to wrap legacy applications for reuse in a SOA.

From the perspective of mapping enterprise services into a given service-oriented business modeling, there is a lack of approaches, though many authors such [14-20] have attempted to classify services into taxonomy.

The drawback of these approaches is that they do not explicitly consider the reuse existing service portfolio of services in their service-oriented BP modeling neither have they reverse engineered the IS into BOs that provide reusable services.

3 Service-Oriented Business Process Modeling

A BP modeling captures the relevant properties with respect to the above-mentioned definition of BPs, BOs and the relationships between them. In our service-oriented modeling approach, two types of BOs provide BPs with specialized services.

3.1 Modeling Concepts

Our modeling uses the concepts of Value, BEs, BOs, BPs, and specialized services, and a set of rules to construct these concepts from the universe of discourse.

Business Objects. To represent BOs in a consistent way regardless of the needs of BP modeling, we insist on the monolithic representation of the activities and data of these BOs so that BOs provide all the required services. Indeed, this representation is not happening nowadays as different representations (or images) of the same BO are developed depending on the needs of each organizational body (e.g., department, business unit). This leads to not only a limited view and inconsistency of BO across the whole organization, but also siloed BOs that not accessible by the organization when data integration is required.

We extend the OMG definition, where a BO is "a representation of a thing active in the business domain, including at least its business name and definition, attributes, behavior, relationships, and constraints", by adding a new type of BO, we refer to as *Business Artifact* that has a name and a state. In this way, BOs provide BPs with activities, data, and state as services.

Real BO represents thing active in the business domain such as customers, accounts, products, bill, or order. It has a name, attributes, behavior, and relationships to other BOs. Figure 1 shows examples of BOs and the relationship of the BOs to the BPs.

Artifact BO is a business artifact, a business uses to coordinate and control a set of activities related to the same BE or value. It has a set of states (including initial and final states) that reflects its execution. Each state has precedence. The modeler sets and changes the states over time when business requirements change.

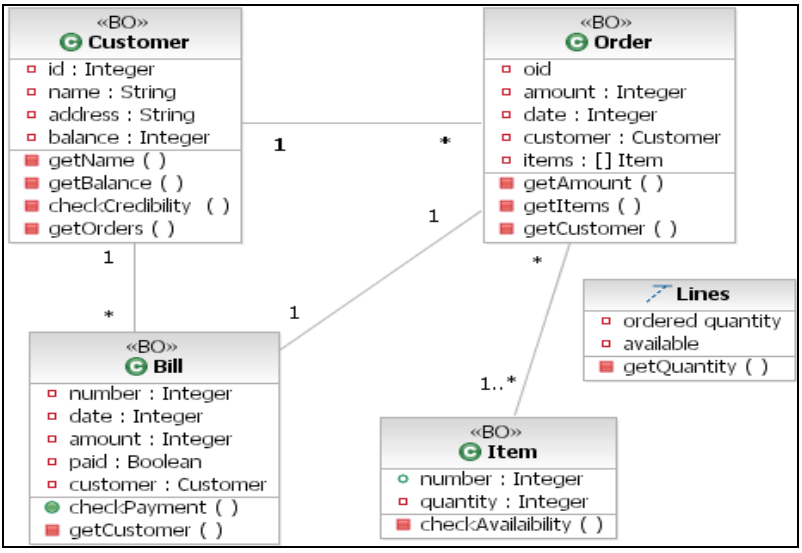


Fig. 1. Examples of BOs

Business Processes. A BP is a description of a service provided as a value to stakeholders upon explicit or implicit demands (of different natures). These demands are expressed by BEs that trigger (initiate) the BPs. A set of coordinated, controlled, synergetic activities realize the service. The BOs provide BPs with these services. A BP has a set of state values (including initial and final state values) that the BP modeler sets and changes over time when business requirements change. A state value reflects the execution BP.

Service. A service [21] may be defined from business and technology perspectives. In our modeling, we use web services and data services.

Our service-oriented approach for BP modeling emphasizes the separation of concerns that differentiate the activities of control and execution. Similarly, the data packaged into real BOs are separated from the state that is packaged in artifact BO.

Accordingly, we specialize services into basic and composite services. The latter is an aggregation of basic services, and has controller service.

- *The Controller Service (CS)* oversees a BP execution through the state of the artifact. The CS deals only with the control and coordination of the BP. It invokes a State Service (SS), provided by the artifact BO, to retrieve the state; and accordingly invokes the respective Worker Service (WW) and updates the state when any of the WS terminates its job. The CS is invoked by an Initiator Web Service (IWS).
- *The Initiator Web Service* deals only with the initialization and starting of an instance of the BP.
- *The State Service* is a data service that represents the structure of the state of the artifact BO. It is used by the CS to retrieve the state of the BP.
- The *Worker Services* add value to a BP towards the achievement of its goal. WKs are provided by real BOs. The CS invokes them according to the state. Inversely, WKs report the outcome of their realization to the CS. WWs can also to retrieve simple or integrate data from BOs.
- WWs may use *Data Services* (DS) if necessary to retrieve simple or integrate data from BOs. DSs retrieve or integrate the requested data from the BOs.
- The following are the related concepts used to model a BP with services and the relationships between them. These are 'use', 'has', or 'is' relationships.

3.2 Relationships between the Specialized Services

There are four types of relationships: association, specialization, realization, and use.

- *Association relationships* indicate how the elements of a BP environment are associated with each other. For instance, a BP is associated with an event, a set of BOs, a set of states, including an initial state, and a final state.
- *Specialization relationships* indicate the specific roles of some elements of the model. For instance, a service may be a CS, a SS, or a WW.
- *Use relationships* show that some services use the capabilities of others. For instance, the CS uses both WWs and SS. The latter provides it with the state of the BP, whereas the former perform the required activity. The IWS uses the CS.
- *Realization relationships* show that WWs are realized in the IS by the BOs, whereas the SS is realized by a specific data structure representing the state values.

4 Towards Service-Oriented Business Process Models from Legacy Information Systems

The drawback of the existing BP modeling approaches is that they do not explicitly consider the legacy IS (LIS) as a provider of services, neither have they included the service portfolio of a company or its partners, in order to have a service-oriented BP modeling in favor of agility. This requires a complete reengineering of both the LIS and the existing BPs (if any) into services mostly realized by the BOs.

We consider the most important scenario, where previous services being supported according to the existing BP descriptions, but are not explicitly described in the services portfolio. This is important for companies that have made a previous BP modeling (a non-service-oriented modeling strategy) and now, they want to move to a service-oriented one, in order to achieve highest enterprise agility. For companies that deal with BP modeling at the first time, this point does not have a lot of sense.

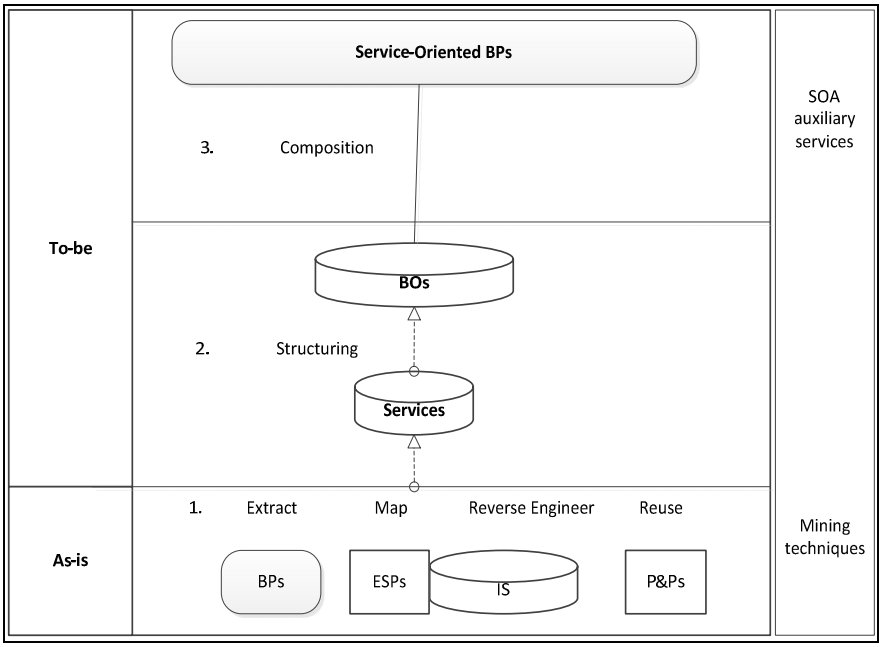


Fig. 2. Techniques to transform as-is BP into service-oriented BP

Moreover, we distinguish between services at technical abstraction level (e.g., web services, data services, or restful services) that conform to the services portfolio, and services at higher abstraction level that could be implicitly supported by BPs in companies. Unfortunately, these abstract descriptions are missing in most companies.

This is challenging, but Model Driven Development can address it by: (i) considering and treating all software artifacts as models that conform to specific meta-models, and (ii) establishing automatic transformations between models at different abstraction levels.

Our approach proposes a road map to transform an as-is BPs into to-be, i.e., the existing software artifacts into basic service and composite service with respect to SOA. We propose to use different techniques as shown in Figure 2, namely (i) reverse engineering IS, including legacy applications and database, and a way to map previous, existing Enterprise Portfolio Services (ESP), (ii) extract services from exiting BPs, or (iii) reuse Partner and Provider Services (P&Ps). In this reverse engineering phase, everything becomes services. Later on, a restructuring phase will categorize these services into CS, SS, or WKs and assign them to their respective real or artifact BOs as shown in Figure 2.

Therefore, our service-oriented BP modeling requires an integrated view of all the reengineering phases [3] i.e., reverse engineering, restructuring and forward engineering. Our aim is to:

1. Make the LIS as service providers, which requires their transformation into real BOs that provide WSs
2. Transform the existing BPs (if any) into artifact BOs that provide SS used by CS

5 Service-Oriented Business Processes by Reverse Engineering

As stated, the reverse engineering phase aims at providing LIS, ESP, existing BP traces, or P&P as services. Whereas, the restructuring phase will categorize these services into CS, SS, or WK and assign them to their respective real or artifact BOs.

5.1 Reverse Engineering LIS to Extract Services

When organizations do not have a portfolio of services, the reverse engineering of legacy IS (LIS) is required to first mine BPs, then extract services from these BPs to further reuse them in service-oriented BP modeling.

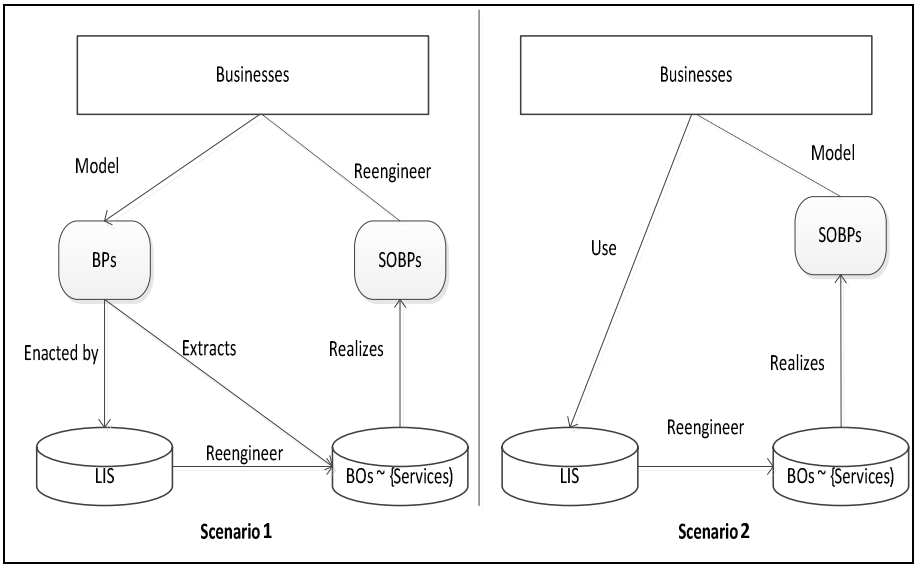


Fig. 3. Scenarios to discover and reconstitute BP embedded in systems

Most BPs in organizations are supported by their IS. The optimal BPM is therefore achieved when organizations additionally combine the management of their LIS [22]. The configuration management of LIS is particularly important since these systems undergo a considerable amount of changes during their lifecycles. Because of the evolutionary maintenance over time, new business knowledge and rules are embedded

in LIS. This embedded business knowledge may not exist anywhere else [23]. The evolution of IS in isolation consequently affects the evolution of BPs (see scenario 2 in Fig. 3). In this case, it is necessary to discover and reconstitute the underlying BPs that are currently supported by LIS [24].

However, there are many organizations that currently carry out a vast amount of daily transactions through their IS without having ever done their own BP modeling. When these organizations deal with BP modeling for the first time, a recurrent method by which to attempt this modeling is the extraction of BP from LIS [25] (see scenario 2 in 3). This is owing to the fact that LIS is one of the few knowledge assets in organizations that can be used to attain an accurate understanding of the actual BP.

In both scenarios (Fig. 3), retrieving an up-to-date version of BPs from LIS allows organizations to take advantage of at least two main benefits:

- Benefits for BP modeling: BPs can always be up-to-date. Organizations may therefore conduct BPM by following the continuous improvement process [26]. This kind of BPM facilitates an agile adaptation of BP to meet the changes that occur in the uncertain environment. The rapid evolution of BPs allows organizations to maintain, or even improve, their degree of competitiveness [27].
- Benefits for LIS: LIS continue to be modernized on more occasions. A recent study by the SEI (Software Engineering Institute) states that it is first necessary to retrieve embedded business knowledge in order to modernize systems in line with the organization BPs [28]. Organizations can thus modernize their LIS whilst they align the new systems with their actual BPs. LIS is therefore evolved rather than being immediately retired and the ROI (return of investment) on such systems is improved.

Moving to service-oriented BP with respect to SOA is possible by taking into account the valuable knowledge embedded in LIS, (including data and applications) for four key reasons: (i) SOA is mainly about reuse of assets, in this regard, LIS are running smoothly and supporting critical tasks, (ii) most of the business functions are locked within LIS [29], (iii) LIS were built at high cost; and we need to preserve these investments [30], and (iv) migration to SOA can give new life to LIS [31].

Yet, the LIS was built without taking into account the advent of service orientation and service-oriented BP, and are incompatible with these environments [32]. Therefore, modernization techniques of these LIS are necessary [12], [33]. However, there exist several modernization techniques, including replacement, redevelopment, migration, reengineering, extension and surrounding, or wrapping [34-36], which requires an understanding of both the LIS and the changes, in business and technology, since these applications were developed.

One of the solutions is to extend the critical business logic of the LIS while preserving the investments, through their wrapping to services, making them loosely coupled, interoperable, discoverable, and (re)usable within and across the boundaries of the enterprise.

A critical aspect of this solution concerns with guidance process that assists IT department teams to select an appropriate solution among several potential, yet confusing, modernization techniques and tools. Indeed, constructing services from LIS to obtain the benefits of service orientation is not an easy task. It requires a complex task, particularly when the services are expected to execute within a tightly con-strained environment [13].

Once we extract a BP from LIS, it is then possible to present it as a set of coordinated activities and data. The BP, its activities and data could easily be mapped into services. Indeed:

- A BP is mapped into CS
- Each activity output of an activity is mapped into state value
- Each activity is mapped into WW
- Each piece of data is mapped into data service

5.2 Reverse Engineering from Enterprise Service Portfolio

The other way around to discover services that comply with our modeling is to use the enterprise service portfolio.

Service Taxonomy. The objective of service mapping is to identify services that are valuable, from business and IT perspectives. However, what constitutes a service has different meanings due to the lack of a standard classification. Several researchers attempted to provide taxonomy [14-20]. Taxonomy supports service-mapping process by clarifying the roles of the different types of services, which helps in understanding the role of each service. It also assists with the discoverability of services, which can further promote reuse. In addition, taxonomy helps to make organizational decisions like how to obtain a capability (build vs. buy vs. lease). To summarize the classification efforts of these researchers, services can be divided into three broad categories:

1. ***Conceptual services*** (also called business services) service represents the core of software product requirements. They express organizational ideas, thoughts, opinions, views, or themes that propose software solutions to organization [17].
2. ***Capability services*** provide an explicit business value, ranging from generic services, reused in multiple service-oriented applications to specialized services, part of a single service-oriented application. Capability services include:
   (a) *Process services* are services whose operations are directed by the BP definition [18][20]. BPEL are example of such kinds of services.
   (b) *Task services* (aka application/activity/capability) encapsulate business logic specific to activities or BP. A task service represents an atomic unit of process logic.
   (c) Entity services (also called data services) represent one or more related business entities such as invoice. Entity service is considered a highly reusable service because it is agnostic to most BPs and workflow [20]. As a result, a single entity service can be reused part of multiple business processes. The type of operations exposed by entity service is typically CRUD (create, read, update, and delete) operations. In addition, entity service may provide the ability to unify and hide differences in the way key data sources represented as databases within the organization.
   (d) *Utility services* contain logic derived from a solution or technology platform. Utility services expose capabilities of multiple applications within the

organization, including migrated, reengineered legacy systems, and some Commercial Of-The-Shelf software (COTS). In addition, utility services represent logic that may not always need to be part of a BP for instance event logging, exception handling [20].

(e) *Hybrid services* contain logic derived from both BPs and applications. Hybrid services expose capabilities wrapped legacy systems and some COTS that may exists within the organization.

(f) *Partner services* are offered to/by external partners based on agreed terms, this type of services is known as Cloud services such as Application as a Service (AaaS) or Software as a Service (SaaS). Partner services are considered as separate type due to security and management concerns. A partner service could present capability offered by any of the application services depending on the organizational requirements.

3. ***Infrastructure services*** are part of the organization supporting distributed computing infrastructure. These are:
   a. *Communication services* transport messages into, out of, and within the system. Examples include publish-subscribe services, queues, and ESB.
   b. *Auxiliary services* provide facilities for monitoring, diagnostics, and management activities of other services. These may include statistical information.

Mapping Services to Business Process. Table 1 shows that any kind of existing services within the organization portfolio (first column) could be mapped into one of the services we use in our modeling: a CS, a WW, or a DS. It is worth noting that none of the ESP is mapped into a SS since SS is specific to our modeling. Infrastructure services map to SOA auxiliary services.

This mapping might be automated if the organization could have a portfolio or services that have common meaning, which is the role of the BOs in our modeling.

Table 1. Services-to-service-oriented BP mapping

Existing (ESPs)	Refined Services	Service-Oriented BP
Conceptual services		CS
Capability services	Process service	CS
	Task service	WW
	Entity service	WW or DS
	Utility service	WW
	Hybrid service	WW
	Partner services	WW
Infrastructure services	Communication services (e.g., ESB)	SOA auxiliary services
	Auxiliary services (e.g., Registry)	

Marching Services with Business Objet Mapping. This transformation between As-Is and To-Be artifact distributes the services we got in the first reverse engineering phase over the real and artifact BOs as shown in Table 2.

Table 2. Services-to-BO mapping

Existing (ESPs)	Refined Services	BOs	
		Real	Artifact
Conceptual services	Conceptual service		√
Capability services	Process service		√
	Task service	√	
	Entity service	√	
	Utility service	√	
	Hybrid service	√	
	Partner services	√	√
Infrastructure services	Communication		
	Auxiliary services (e.g., Registry)		

6 Conclusions

We have provided guidance towards retrieving services towards service-oriented business process modeling, from legacy information systems.

First, we have provided a new modeling for business processes, where the elements of business process environment are modeled as services, including controller service, state service and worker services. The worker services and state service are provided by real and artifact business objects respectively.

Then, we have shown that this guidance uses different techniques to move from as-is business processes to to-be business processes. We have categorized these techniques into: (i) reverse engineering the legacy information, by extracting services from traces of running business processes, and (ii) reverse engineering from the enterprise service portfolio and reusing partner and provider services system.

Finally, we have sketched out the challenges and the process of each technique.

This is a step towards moving SOA maturity towards the next levels, where services are part of the requirements and the solutions as business-related services and IT-related services respectively. This would promote integration, composition, flexibility, and agility in response to changing business events.

Although, we have presented approaches and processes for transformation techniques having a real impact on the way business processes should responsive to the changes in the business requirements by using web service-based SOA, this work has limitation. Therefore, we would consider that this work has presented the service-oriented business process as rather a roadmap towards research issues and questions related to transformation techniques than a definitive solution.

We need to develop tools and dig deeper in the different transformation techniques.

Further work would complete the cycle with the forward engineering phase by using MDD, as services define both the requirements and the solutions.

References

1. Baghdadi, Y.: Modelling business process with services: Towards agile enterprise. Int. Journal of Business Information Systems (in press, 2013)
2. Cummins, F.A.: Building the Agile Enterprise: With SOA, BPM and MBM. Morgan Kaufmann, San Francisco (2010)

3. Chikofsky, E.J., Cross, J.H.: Reverse engineering and design recovery: a taxonomy. IEEE Software 7(1), 13–17 (1990)
4. Indulska, M., Recker, J., Rosemann, M., Green, P.: Business Process Modeling: Current Issues and Future Challenges. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 501–514. Springer, Heidelberg (2009)
5. Baghdadi, Y.: A business model for B2B integration through Web services. In: IEEE Int. Conference on e-Commerce Technology, pp. 187–194. IEEE (2004)
6. Cauvet, C., Guezilian, J.: Business Process Modeling: a Service-Oriented Approach. In: Hawaii 41st Annual Int. Conference on System Sciences, pp. 1–8. IEEE (2008)
7. Stein, S.: Modelling Method Extension for Service-Oriented Business Process Management. PhD diss., Kiel, Christian-Albrechts-Universität, Diss. (2010)
8. Rahgozar, M., Oroumchian, F.: An effective strategy for legacy systems evolution. J. of Software Maintenance and Evolution: Research and Practice 15(5), 325–344 (2003)
9. Pérez-Castillo, R., de Guzmán, I.G.-R., Piattini, M.: Business process archeology using MARBLE. Information and Software Technology 53(10), 1023–1044 (2011)
10. Khadka, R., Saeidi, A., Idu, A. Hage, J., Jansen, S.: Legacy to SOA Evolution: Evolution: A Systematic Literature Review. Technical Report UU-CS-2012-006 (2012)
11. Baghdadi, Y.: Reverse engineering relational databases to identify and specify basic Web services with respect to service oriented computing. Information Systems Frontiers 8(5), 395–410 (2006)
12. Sneed, H.M., Schedl, S., Sneed, S.H.: Linking legacy services to the business process model. In: 6th IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), pp. 17–26. IEEE (2012)
13. Baghdadi, Y., Al-Bulushi, W.: A Guidance process to modernize legacy applications for SOA. Service Oriented Computing and Applications, Online First Articles (2013)
14. Al-Rawahi, N., Baghdadi, Y.: Approaches to identify and develop Web services as instance of SOA architectures. In: Int. Conference on Services Systems and Services Management (ICSSSM 2005), pp. 579–584. IEEE (2005)
15. Gu, Q., Lago, P.: Service Identification Methods: A Systematic Literature Review. In: Di Nitto, E., Yahyapour, R. (eds.) ServiceWave 2010. LNCS, vol. 6481, pp. 37–50. Springer, Heidelberg (2010)
16. Lago, P., Razavian, M.: A Pragmatic Approach for Analysis and Design of Service Inventories. In: Pallis, G., et al. (eds.) ICSOC 2011 Workshops. LNCS, vol. 7221, pp. 44–53. Springer, Heidelberg (2012)
17. Marks, E.A., Bell, M.: Executive's Guide to Service-Oriented Architecture. John Wiley & Sons (2006)
18. Cohen, S.: Ontology and taxonomy of services in a service-oriented architecture. The Architecture Journal 11, 30–35 (2007)
19. Cho, M.J., Choi, H.R., Kim, H.S., Hong, S.G., Keceli, Y., Park, J.: Service Identification and Modeling for Service Oriented Architecture Applications. In: 7th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems, pp. 193–199. WSEAS (2008)
20. Erl, T., Taub, M.L., Hart, K., Mcfarland, J., Young, T.: SOA Design Patterns. Prentice Hall (2009)
21. Chesbrough, H., Spohrer, J.: A research manifesto for services science. Communications of the ACM 49(7), 35–35 (2006)
22. Jeston, J., Nelis, J.: Business process management. Elsevier Publisher (2012)
23. Paradauskas, B., Laurikaitis, B., Business, A.: knowledge extraction from legacy information systems. Information Technolgy and Control 35(3), 214–221 (2006)

24. Van den Heuvel, W.J.: Aligning Modern Business Processes and Legacy Systems: A Component-based Perspective. The MIT Press (2009)
25. Van der Aalst, W., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., Alves de Medeiros, A.K., Song, M., VErbeek, H.M.W.: Business process mining: An industrial application. Information Systems 32(5), 713–732 (2007)
26. Davenport, T.H.: Need radical innovation and continuous improvement? Integrate process reengineering and TQM. Strategy & Leadership 21(3), 6–12 (1993)
27. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2012)
28. Lewis, G.A., Smith, D.B.: A Research Agenda for Service-Oriented Architecture): Maintenance and Evolution of Service-Oriented Systems, Technical Note, CMU/SEI-2010-TN-003 (2010)
29. Galinium, M., Shabaz, N.: Success factors model: Case studies in themigration of legacy systems to service-oriented architecture. In: Int. Joint Conference on Computer Science and Software Engineering (JCSSE), pp. 236–241 (2012)
30. Linthicum, D.S.: Leveraging SOA and legacy systems. Business Integration Journal, Legacy Integration Supplement (2004)
31. Bhallamudi, P., Telly, S.: SOA migration case studies. In: IEEE Int. Conference on Systems (SysCon), pp. 123–128. IEEE (2011)
32. Chenghao, G., Min, W., Xiaoming, Z.: A wrapping approach and tool for migrating legacy components to Web services. In: 1st Int. Conference on Networking and Distributed Computing (ICDNC), pp. 94–98. ICDNC (2010)
33. Lewis, G.A., Morris, E.J., Smith, D.B., Simanta, S.: Smart: Analyzing the reuse potential of legacy components in a service-oriented architecture environment. Technical Note, CMU/SEI-2010-TN-003 (2008)
34. Comella-Dorda, S., Wallnau, K., Seacord, R.C., Robert, J.: A Survey of Legacy System Modernization Approaches', Carnegie Mellon University, Tech. Note, CMU/SEI-2000-TN-003 (2000)
35. Canfora, G., Fasolina, A.R., Frattolillo, G., Tramontana, P.: A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. J. of Systems and Software 81, 463–480 (2008)
36. Umar, A., Zordan, A.: Reengineering for service oriented architectures: A strategic decision model for integration versus migration. J. of Systems and Software 82(3), 448–462 (2009)

Automated Generation of Performance Test Cases from Functional Tests for Web Applications

Federo Toledo Rodríguez[1], Matías Reina[1], Fabián Baptista[1], Macario Polo Usaola[2], and Beatriz Pérez Lamancha[2]

[1] Abstracta, Montevideo, Uruguay
{ftoledo,mreina,fbaptista}@abstracta.com.uy
[2] Universidad de Castilla-La Mancha, Ciudad Real, Spain
{macario.polo,beatriz.plamancha}@uclm.es

Abstract. When modernizing systems there are big risks concerning functional and non-functional properties. It is expected that the functionality, the performance and the dependability are the same (or better) in the new version. Therefore, the preventive workload simulation (to verify non-functional properties) is crucial to guarantee the success of the modernization project. Since tools for load simulation work at protocol level, the automation of tasks for workload simulation demand much more effort than functional testing, whose test cases are designed using record and playback techniques on the GUI: these tools are more intuitive and they have to handle less variables and technical issues. In this article we present a tool to automatically generate workload simulation scripts from automated functional tests. The tool has been used in several projects in the industry, achieving important cost savings and improving flexibility when verifying non-functional properties of a migrated system.

Keywords: Software Testing, Information System Testing, Testing Automation, Non-functional Testing.

1 Introduction

In order to reduce risks, both the functional and non-functional properties of systems must be verified before its deployment into the production environment: in fact, functional and non-functional properties are essential for the success of the deployment and the user acceptance. Typically, tests are performed at different levels to verify and improve system functionalities: unit, component, integration or system testing. Project development is generally iterative with several product releases during its lifecycle, sometimes because they were previously planned, and others ought to bug fixes or other kind of required maintenance interventions. With each new release, test cases corresponding to previous versions are newly executed against the system. The goal of this "regression test" is to find the possible faults that have been introduced between two system versions. Different tools are available to automate the execution of these tests [1]. Functional test cases simulate interactions of the user against the system and are usually implemented as test scripts that are executed by "capture and replay tools". Non-functional tests also consist of test scripts that are launched against

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 164–173, 2013.

the system under test (the SUT), but they are described at a much more low level, what includes of the communication protocol. In non-functional tests, the range of measurable properties is very wide. In this paper we pay special attention to the conditions that must be verified when the system is under the concurrence of multiple users, as for example performance and dependability. Therefore, load simulation tools are used to concurrently generate hundreds of users connected to the SUT [2]. When the load is simulated, the infrastructure experts analyze the health status of the system, looking for bottlenecks and improvement opportunities.

As we will show later, traditional workload simulation approaches have important drawbacks that make its automation very costly and demanding. Also, the resulting testing artifacts are very fragile, in the sense that they are very susceptible to changes in the SUT: modifications in the SUT (even bug fixes) often require the adaptation and maintenance of test cases for the next stage of regression testing.

Since functional test automation is much easier than the automation for workload simulation (what includes its easiness to be maintained and understood), our proposal is to take advantage of the functional test scripts to automatically generate workload simulation scripts. Focused on web systems, the idea consists automatically in executing the functional test scripts while the HTTP traffic is captured. Later, the HTTP trace is analyzed to generate a workload simulation script model which is finally used to generate the script code to be executed by a load generator.

The rest of the article is organized as follows: section 2 goes deeper on automation of functional tests (regression tests) and workload simulation, especially to measure performance for web systems; section 3 presents the proposal that is then validated in section 4, showing the first results for the usage of the tool in the industry; and after mentioning the related work in section 5, the conclusions and future work is presented in section 6.

2 Background

An extended and current practice in the development of web systems is the **automation of functional tests**, using tools to simulate the user's actions, like Selenium (seleniumhq.org) or WatiN (watin.org), just to mention some of the most popular open source projects. This kind of tools offers the possibility to follow a *record-and-playback* approach. Basically, it is necessary to manually execute the test case while the tool captures every action performed by the user on the browser. Then, the tool stores the actions in a file with a specific format or language (the *test script*) that the same tool can reproduce later. The same test case can be executed as many times as needed, performing the defined validations. Every command of the test script simulates a user action.

Command	Target	Value
open	/sampleApplication/home.aspx	
clickAndWait	link=Search	
type	id=vSEARCHQUERY	computer
click	name=BUTTON1	

Fig. 1. Selenium script for functional testing

These commands take as parameters the HTML elements on which the captured action has been executed (for example, the input entered in a form), and the values entered. Fig. 1 shows an excerpt of a Selenium test script that accesses to an application (1st line), clicks the "Search" link (2nd line), enters the "computer" value in the HTML field "vSEARCHQUERY" (3rd line), and finishes by clicking the button with name "BUTTON1" (4th line).

This schema, in most of the tools, can be complemented with a *data-driven testing* approach, by which the test case takes test data from a separated source (a text file or a database, called "data pool"). Therefore, the same script can be reproduced with different data sets, testing in that way different cases with just little extra effort: adding lines to the test data source.

A **workload simulation** (also known as **load test** or **performance test**) could be defined as a technical research to determine or validate the velocity, scalability and stability characteristics of a SUT, in order to analyze its performance under load conditions [2]. This kind of tests are useful to reduce risks towards the *going live* day, analyzing and improving the non-functional aspects of the application and the performance of the different servers when they are exposed to the concurrent users [3, 4]. There are specific tools to do that, called *load generators* or *load testing tools*, simulating concurrent users accessing to the system. Two of the most popular *open source* load generators are OpenSTA (opensta.org) and JMeter (jmeter.apache.org).

Unlike the functional test automation, in the workload scripts, even though the *record and playback* approach is also common, the tools do not record at a graphic user interface level. Instead, they do it at the communication protocol level. This happens because a functional test script reproduces the user actions on a real browser, whilst load generators try to "save" resources doing the simulation at a protocol level: for the HTTP protocol, for example, the tool will launch multiple processes that just send and receive the corresponding byte arrays through a network connection. Since the goal of these tests is to check the behavior of the server, neither the user interface nor other kind of graphic elements are required.

The workload script contains a sequence of commands that manage HTTP requests and responses according to the protocol. This script is much more complex than the equivalent functional test script. For example, for the same actions presented in Fig. 1, where the script has only four lines of Selenium code, the equivalent performance test script has 848 lines using OpenSTA. That corresponds to each HTTP request sent to the server: take into account that each request triggers a sequence of secondary requests, which correspond to images included in the webpage, CSS files, Javascripts, etc. Each request (primary or secondary) is composed by a header and a body, as shown in the example of Fig. 2. Embedded, there are parameters, cookies, session variables, and any kind of elements used in the communication with the server. The example in this figure corresponds to the primary HTTP request of the last step of the test case; so, it includes the value "computer" in the parameter "vSEARCHQUERY" (the box).

Once the script is recorded, a number of adjustments must be performed in order to make it completely reproducible and representative of real users: for example, it has no sense to execute all the test cases with the same user name and password, the same search key (because of caches), etc. These scripts will be executed by concurrent processes (known as *virtual users*). The cost of the needed adjustments depends on

the automation tool and on the SUT. In most of the cases, it is necessary to adjust the management of cookies and session variables (many of them must be unique or fulfill other restrictions). Adjustment of parameters in the header and in the body will also be required.

```
POST URI "http://localhost/sampleapplication/search.aspx HTTP/1.1" ON 1 &
    HEADER DEFAULT_HEADERS    &
    ,WITH {"Accept: */*", &
        "Accept-Language: en-US", &
        "Referer: http://localhost/sampleapplication/search.aspx", &
        "Cookie: "+S cookie 1 0+"; "+S cookie 1 1}        &
,BODY "vSEARCHQUERY=computer&BUTTON1=Search&GXState=%7B%22_EventName%" &
    "22%3A%22E~<27>SEARCH~<27>.%22%2C%22_EventGridId%22%3A45%2C%22_Ev" &
```

Fig. 2. OpenSTA script for performance test

We have analyzed more than 20 projects where we prepared a workload simulation in real performance testing projects, reporting that the scripting phase takes between 30% and 50% of the total invested effort. On the other hand, the maintenance of these scripts (when the SUT changes) tends to be so complex that it is better to rebuild a script from scratch instead of trying to adjust it. Because of that, the process becomes pragmatically inflexible. The test generally will identify improvement opportunities, which imply modification on the system; however, our scripts will stop working if we change the system. How can we verify that the changes take a positive effect?

3 Automatic Generation of Workload Simulation Scripts

The methodology proposed extends the automation phase of the process presented in Vázquez et al., [3]. Instead of building the workload simulation scripts from scratch, the user has to provide a set of automated functional test. As shown in Fig. 3, our tool, which is a module of our testing framework called GXtest (gxtest.abstracta.com.uy), will build a model of the HTTP traffic captured from the execution of each of these scripts. The model is the entry of the tool that generates the script code for the preferred load testing tool.

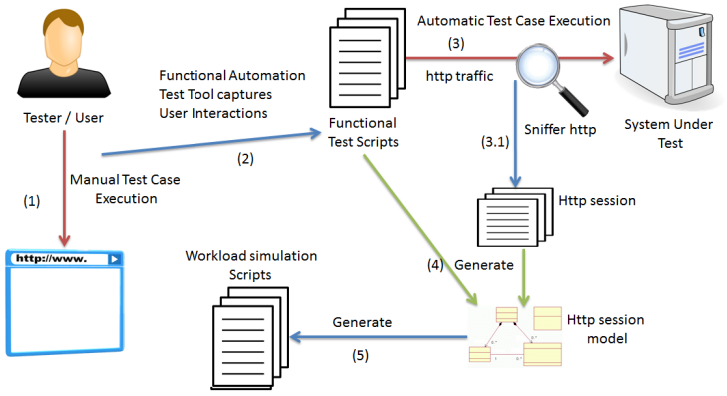


Fig. 3. Scripts Generation proposal for Workload Simulation Tests

GXtest executes Selenium and WatiN scripts, but it can be easily extended for more automated testing tools. During the execution of the functional test scripts, it captures the HTTP traffic between the browser and the SUT with an HTTP sniffer (a tool capable of capturing the network traffic) called Fiddler (fiddler2.com). With this information it builds a model that is used to generate the scripts for OpenSTA or JMeter. Also, it is easily extensible to generate scripts for other load simulation tools.

Fig. 4 shows the main elements of the HTTP traffic model, which is useful to generate the workload simulation scripts. This model is built using the information obtained by the sniffer (all the HTTP requests and responses) and by the functional test scripts, correlating the user actions with the corresponding HTTP traffic. It is therefore composed of an ordered sequence of actions, including invocations to the application through HTTP (*requests*), or *validations* on the response to verify that it is as expected. Each HTTP request is composed of a *header* and a message *body*. Both parts of the message are composed of parameters with their corresponding values. The header also has a set of fields that include, among others, cookies and session data. Each value can be hardcoded or can be taken from a data pool. It is important to keep the references between each HTTP request and its response, and with the corresponding functional test script command that generated it.

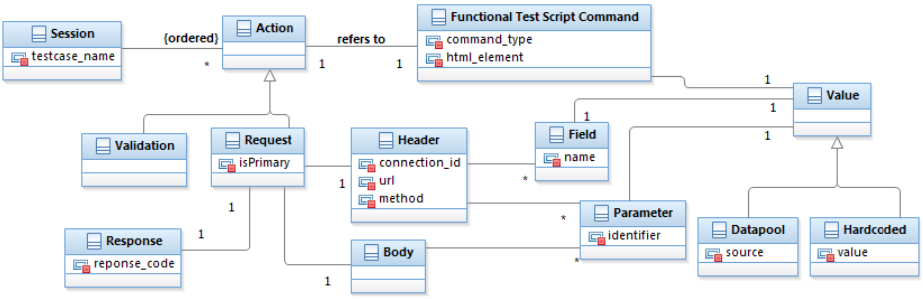


Fig. 4. HTTP Session Metamodel

This model is used to generate code according to the language provided by the load generation tool. The generated code is specifically for OpenSTA or JMeter, according with the user's preferences for the workload simulation. To perform this code generation the tool has an approach similar to the one proposed in model-driven environments for the model-to-text transformations [5], where the code generation is defined with code templates for each element of the model. Table 1 includes some examples of those templates for OpenSTA; the first one is for the general structure of the script, used for each test case of the model, and the second one corresponds to an HTTP request, according to the specification of the HTTP protocol.

As mentioned, the resulting script must be adjusted after the recording. Many of them are very repetitive tasks. Our tool makes this kind of things automatically, using the templates mechanism. Some of them are:

- Adding *timers* to each user action in order to measure the response time when executing the test scenarios, considering the kind of actions performed in the functional test script and the corresponding HTTP requests for each one.

- Taking advantage of different design aspects of the functional test script, in the performance test scripts: (1) the data are taken from the same data pools; (2) the same validations are performed; (3) same structure and modularization in different files promoting the readability of the test script.

In this way we get scripts even better than when recording them with the OpenSTA or JMeter recorders. The more we use the tool, the more improvements and automatic adjustments we add to the scripts, avoiding that the tester commits mistakes during the preparation of the performance test.

Once the scripts are finished, the effort can be invested in the most important part (and the most interesting and beneficial) of a performance testing project: the execution of the load scenario and the system's behavior analysis.

The approach is implemented as a module of the commercial tool GXtest, and it is available in www.abstracta.com.uy.

Table 1. Templates for scripts generation

```
[template public generateScript(s: Session)]
[file (s.testcase_name().concat('.scl'), false, 'UTF-8')]
Definitions
        Timer T_TestCase_[s.testcase_name/]
        [s.variableDeclarations()/]
        CONSTANT DEFAULT_HEADERS = "Host:  [s.getBaseURL()/]
        User-Agent: Mozilla/4.0"
Code
        Entry USER_AGENT,USE_PAGE_TIMERS
        Start Timer T_TestCase_[s.testcase_name/]
        [s.processActions()/]
        End Timer T_TestCase_[s.testcase_name/]
Exit
[/file]
[/template]
```

```
[template public processRequest(r: Request)]
Start Timer [r.name/]
[if ([r.isPrimary/])]PRIMARY [/if] [r.header.method/] URI [r.header.url/]
HTTP/1.1" ON [r.header.connection_id/] &
        HEADER DEFAULT_HEADERS, WITH [r.header.processFields()/]}
        [r.processBody()/]
[r.response.processLoadCookies()/]
End Timer [r.name/]
[/template]
```

4 First Experiences in the Industrial Usage of the Tool

In the moment of writing these lines, the presented tool has been used in five different projects of five different customers of Abstracta. Abstracta is a Uruguayan company specialized in providing testing services. The tool we are presenting here has been developed in its R+D department, in collaboration with the University of Castilla-La Mancha, in Spain.

There were two testers working in all the projects, both with high knowledge about Selenium and OpenSTA. The SUTs were web systems from different domains and

developed with different technologies, and very good results were obtained in all of them. Table 2 shows the number of generated scripts for each project, and the amount of simulated virtual users concurrently accessing to the SUT, and the amount of PCs required for the execution of the workload simulation.

Table 2. Use of the tool in performance testing projects

Project	SUT	# Scripts	# VU	# PCs
Human Resources System	AS400 database, Java Web system on Websphere	14	317	1
Production Management System	AS400 database, C# Web system on Internet Information Services	5	55	1
Courts Management System	Java Web system on Tomcat with Oracle database	5	144	1
Auction System	Java Web system on Tomcat with MySQL database	1	2000	4
Logisitcs System	Java Web system on Weblogic with Oracle database	9	117	1

It is important to highlight that there are cases with few scripts, like in the 4th row, where only one script was required. That was defined based on the statistical analysis about the normal use of the system, which revealed that the 80% of the load is generated only with few use cases. However, the combination of this test script with the test values from data pool leads to different execution flows in the SUT.

In some projects the SUT was developed with Model-driven Development tools (particularly with GeneXus: www.genexus.com) capable of generating code from models with a higher level of abstraction. This raises a special complication, because just small modifications to the models could mean many modifications on the generated code and therefore on the HTTP traffic. The process was the same as in the rest of the systems that were tested: first it is necessary to adjust the functional test scripts to regenerate the workload simulation scripts with our tool. It is in this kind of systems, where the SUT suffers many modifications during the testing project, where our approach reports the best benefits, because it was necessary to regenerate the scripts several times, and this would have required a major effort if manually executed.

In one of the projects there were no previous functional test scripts, so it was necessary to automate functional test scripts to use the tool. These functional test scripts were developed by a user (without knowledge about regression testing) which is almost impossible with any load generator. Once the project ended, the testing team started to manage a regression testing environment, using the scripts that were developed in the performance test project. In a certain way, the performance quality control favored the functionality quality control.

We also show in the table the amount of PCs required for the execution of the workload simulation in order to highlight that, with our approach, we can execute the simulation with a reduced test infrastructure.

The main limitation of the tool is that it only supports HTTP protocol. We believe that the approach is valid to any client-server architecture, and we are working in that direction, but we started with web systems because the migration to this platform has been very common and risky at the same time. Also, even restricting to web systems, there are different technologies to develop web systems, each one with its

particularities. The different systems we have tested have shown us that each new technology we test implies some adaptations to our templates to generate the workload simulation scripts (for example, it is not the same a simple PHP system than one that makes use of Ajax). However, once the template is adjusted for certain technology, it can be used for any system implemented with it.

Even considering the limitations, the time invested in the preparation of the scripts was reduced from an average between 6 to 10 hours in the traditional approach (in our previous projects) to 1 to 5 hours with our tool in these five case studies. That implies an important cost saving for the automation phase. In addition, as it is easier to regenerate the scripts, it also gave us more flexibility for the maintenance of the scripts.

To summarize, the case studies have shown promising results in the performance testing, demonstrating that it can be made in a more flexible way and with less effort, according with what the testers involved in the projects reported. These results are also aligned with the ones reported in the case study of [6].

5 Related Work

There are tools that, in order to ease the construction and maintenance of the workload simulation scripts, work at a graphic user interface, using Selenium scripts to execute load tests. The limitation of this approach comes from the fact that using PCs is probably not enough to simulate the typical number of users of a load test. These tools typically execute the tests scenarios from the Cloud, or with huge infrastructures. Some examples are Scaleborn (www.scaleborn.com) and Test Maker (www.pushtotest.com). With our approach instead, the number of required machines is kept low, being in that way a cheaper alternative, and obtaining the same results. Also, we have the requirement from some companies that they want to use only their infrastructure to execute their tests because of confidentiality issues, then, the solutions in the Cloud cannot be applied. As we showed before, our proposal requires only few machines to execute the workload simulation, and it can be done *in situ*.

As far as we know, there are few proposals to generate performance tests. Some propose to design models as the basis of the performance test scripts generation, as in the one published by García-Domínguez et al. [7], which points to performance testing of workflows systems invoking Web Services. Instead, our proposal is for web systems.

There are also some proposals to use stereotyped UML models, such as [8–10], or even others that extend a UML design tool to generate a complete set of performance test artifacts from the modeled diagrams [11]. The main disadvantage of these proposals is that a big effort is required to design the input artifacts for the generator. In our case the user has to specify the test cases in the same simple way it is done in functional testing.

Last but not least, we would like to highlight the article of de Sousa et al. [6] where a similar approach is proposed, taking advantage of the functional test scripts to generate workload simulation scripts. They propose a syntactical transformation, translating Selenium commands to JMeter commands. We observe two important limitations with this approach: on the one hand, as they do not consider the HTTP traffic (they only use the functional test script as input), it is impossible to generate the secondary

requests and the primary requests coming from redirects that the SUT is doing, and on the other hand, it is not considering any javascript modification on the requests; therefore, the resulting simulation is not faithful to the real users load. This is why in our approach we decided to execute the functional script in order to have confidence to be executing the correct traffic that a real user would execute.

6 Conclusions and Future Work

In order to reduce risk releasing a system to the final users, workload simulation testing is needed to validate non-functional properties such as performance and dependability. As this task is expensive and resource demanding, it is typically made in a poor or incomplete way, or the results come too late. The most demanding task is the automation of the functionalities to be tested, taking part of the time that could be used to execute tests and analyze how to improve the system.

Taking this into account, this article presented a new module of the GXtest framework, where the novel contributions are:

- Easier construction of workload simulation scripts: they are generated from automated functional test cases.
- Major flexibility when adjusting test scripts according to the changes and improvements performed on the application. It is only necessary to adjust the functional test cases and regenerate the workload simulation scripts.
- Automatic generation of better scripts, with better quality, in less time and with less effort. The changes to be done to each script are systematized by the tool.

GXtest has being used in different projects to test the performance of a variety of systems, demonstrating the benefits of the proposal.

One of the future work lines is related to the protocols used to execute the workload. JMeter supports different communication protocols, allowing the execution of tests against systems that are accessed by different interfaces (HTTP, SOAP, FTP), and managing the test centralized in one single tool. Therefore, GXtest can be extended to other protocols even generating for the same load simulation tool.

On the other hand, we are also researching in a complete model-driven approach, starting from the requirements to generate functional test cases, and then use this test model to generate automated test cases that are useful to generate workload simulation scripts. Also, from the non-functional requirements it is possible to automatically generate the load scenarios to be executed by the load simulation tool (in this paper we presented how to create executable test cases, but not how to combine them in order to generate the workload), and the non-functional validations to verify that the requirements are being reached.

Acknowledgements. This work has been partially funded by *Agencia Nacional de Investigación e Innovación* (ANII, Uruguay) and by the GEODAS-BC project (TIN2012-37493-C03-01). We would also like to express our special acknowledgement to Abstracta team.

References

1. Graham, D., Fewster, M.: Experiences of Test Automation: Case Studies of Software Test Automation. Addison-Wesley Professional (2012)
2. Meier, J., Farre, C., Bansode, P., Barber, S., Rea, D.: Performance testing guidance for web applications: patterns & practices. Microsoft Press (2007)
3. Vázquez, G., Reina, M., Toledo, F., de Uvarow, S., Greisin, E., López, H.: Metodología de Pruebas de Performance. Presented at the JCC (2008)
4. Barber, S.: User Experience, not Metrics (2001)
5. OMG, MOF Model to Text Transformation Language (MOFM2T), 1.0 (2008)
6. de Sousa Santos, I., Santos, A.R., de Alcantara dos, P., Neto, S.: Reusing Functional Testing in order to Decrease Performance and Stress Testing Costs. In: SEKE, pp. 470–474 (2011)
7. García-Domínguez, A., Medina-Bulo, I., Marcos-Bárcena, M.: Performance Test Case Generation for Java and WSDL-based Web Services from MARTE. Adv. Internet Technol. 5(3&4), 173–185 (2012)
8. Garousi, V., Briand, L.C., Labiche, Y.: Traffic-aware stress testing of distributed sys-tems based on UML models. In: ICSE, New York, NY, USA, pp. 391–400 (2006)
9. Shams, M., Krishnamurthy, D., Far, B.: A model-based approach for testing the performance of web applications. Presented at the SOQUA, 54–61 (2006)
10. da Silveira, M., Rodrigues, E., Zorzo, A., Costa, L., Vieira, H., Oliveira, F.: Generation of Scripts for Performance Testing Based on UML Models. In: SEKE, pp. 258–263 (2011)
11. Cai, Y., Grundy, J., Hosking, J.: Experiences Integrating and Scaling a Performance Test Bed Generator with an Open Source CASE Tool. In: Presented at the ASE, pp. 36–45 (2004)

Investigating the Applicability of the Laws of Software Evolution: A Metrics Based Study

Nicholas Drouin and Mourad Badri

Software Engineering Research Laboratory, Department of Mathematics and Computer Science, University of Quebec, Trois-Rivières, Québec, G9A 5H7, Canada
{Nicholas.Drouin,Mourad.Badri}@uqtr.ca

Abstract. The study aims at investigating empirically the applicability of Lehman's laws of software evolution. We used a synthetic metric (*Quality Assurance Indicator*), which captures in an integrated way various object-oriented software attributes. The goal was to explore if the metric can be used to support the applicability of Lehman's laws of software evolution. We focused on five laws: *continuing change, increasing complexity, self-regulation, continuing growth* and *declining quality*. We performed an empirical analysis using historical data collected on two open source (Java) software systems. Empirical results provide evidence that the considered Lehman's laws are supported by the collected data and the metric.

Keywords: Software Evolution, Laws of Software Evolution, Software Quality, Software Attributes, Metrics, Quality Assurance.

1 Introduction

Software systems need to continually evolve during their life cycle for various reasons: adding new features to satisfy user requirements, changing business needs, introducing novel technologies, correcting faults, improving quality, etc. [1, 2]. The accumulation of changes, along the evolution of a software system, can lead to a degradation of its quality [3-7]. It is, therefore, important to monitor how software quality evolves so that quality assurance (QA) activities can be properly planned [7]. Software evolution is, in fact, the dynamic behavior of programming systems as they are maintained and enhanced over their lifetimes [8]. Lehman's laws of software evolution [4, 5] state that for keeping software systems long-lived continuous change is required. The laws also suggest that due to changes and growth over time, software systems become more complex and it becomes more and more difficult to extend them by adding new functionalities. Software metrics can be used to analyze the evolution of software systems [9]. Metrics have, in fact, a number of interesting characteristics for providing evolution support [10]. A large number of metrics have been proposed for measuring various properties of object-oriented (OO) software systems [11]. Empirical evidence exists showing that there exists a relationship between (many of) these metrics and software quality attributes [9, 12-21]. However, with the growing complexity and size of OO software systems, the ability to reason about such a major issue using synthetic metrics would be more appropriate in practice.

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 174–189, 2013.

We proposed in [22] a new metric, called *Quality Assurance Indicator* (Qi), which captures in an integrated way the interactions between classes and the distribution of the control flow in a software system. The *Quality Assurance Indicator* of a class is based on intrinsic characteristics of the class, as well as on the *Quality Assurance Indicator* of its collaborating classes. It is important to notice, however, that the metric has no ambition to capture the overall quality of OO software systems. Furthermore, the objective is not to evaluate a design by giving absolute values, but more relative values that may be used for identifying critical classes on which more QA effort is needed to ensure software quality. In [22], we performed an empirical analysis using data collected from several open source (Java) software systems. In all, more than 4,000 classes were analyzed (400 000 lines of code). We compared the Qi metric, using the Principal Components Analysis (PCA) method, to various well known OO metrics. The selected metrics were grouped in five categories: coupling, cohesion, inheritance, complexity and size. Empirical results provide evidence that the Qi metric captures, in a large part, the information provided by the studied OO metrics. Moreover, we explored in [12] the relationship between the Qi metric and testability of classes and investigated in [23] the capacity of the Qi metric in predicting the unit testing effort of classes using regression analysis. Results provide evidence that the Qi metric is able to accurately predict the unit testing effort of classes. More recently, we explored in [24] if the Qi metric can be used to observe how quality, measured in terms of defects, evolves in the presence of changes and in [25] if the Qi metric captures the evolution of two important OO metrics (related to coupling and complexity).

In this paper, we wanted to investigate thoroughly if the Qi metric, as a synthetic metric, can be used to support the applicability of Lehman's laws of software evolution [4, 5]. We focused on five laws: *continuing change, increasing complexity, self-regulation, continuing growth and declining quality*. We addressed software evolution from both software internal and external perspectives. We performed an empirical analysis using historical data collected on two open source (Java) software systems. The collected data cover a period of more than four years (fifty-two versions) for the first system and more than seven years (thirty-one versions) for the second one. Empirical results provide evidence that the considered Lehman's laws are supported by the collected data and the Qi metric.

The rest of this paper is organized as follows: Section 2 gives a survey on related work. The Qi metric is introduced in Section 3. Section 4 presents the empirical study we performed. Finally, Section 5 concludes the paper.

2 Related Work

Mens et al. [10] provide an overview of the ways software metrics have been (and can be) used to analyze software evolution. A distinction is made between using software metrics before the evolution has occurred (*predictive*) and after the evolution has occurred (*retrospective*). To support retrospective analysis, metrics can be used to understand the quality evolution of a software system by considering its successive releases. In particular, metrics can be used to measure whether the quality of a software has improved or degraded between two releases. Dagpinar et al. [15] investigate the significance of different OO metrics for the purpose of predicting maintainability

of software. Nagappan et al. [26] focus on mining metrics to predict component failures. The authors noted that there is not a single set of complexity metrics that could be used as a universally best defect predictor. Ambu et al. [27] address the evolution of quality metrics in an agile/distributed project and investigate how the distribution of the development team has impacted the quality of the code.

Lee et al. [9] provide an overview of open source software evolution with software metrics. The authors explored the evolution of an open source software system in terms of size, coupling and cohesion, and discussed its quality change based on the Lehman's laws of evolution [4, 5, 28]. Jermakovics et al. [29] propose an approach to visually identify software evolution patterns related to requirements. Mens et al. [30] present a metrics-based study of the evolution of Eclipse. The authors consider seven major releases and investigate whether three of the laws of software evolution (continuing change, increasing complexity and continuing growth) were supported by the data collected. Xie et al. [1] conduct an empirical analysis on the evolution of seven open source programs and investigate also on Lehman's laws of software evolution. Murgia et al. [18] address software quality evolution in open source projects using agile practices. The authors used a set of OO metrics to study software evolution and its relationship with bug distribution. According to the achieved results, Murgia et *al.* concluded also that there is not a single metric that is able to explain the bug distribution during the evolution of the analyzed systems. Zhang et al. [7] use c-charts and patterns to monitor quality evolution over a long period of time. The number of defects was used as a quality indicator. Eski et al. [16] present an empirical study on the relationship between OO metrics and changes in software. The authors analyze modifications in software across the historical sequence of open source projects and propose a metrics-based approach to predict change-prone classes. Yu et al. [31] study the possibility of using the number of bug reports as a software quality measure. Using statistical methods, the authors analyze the correlation between the number of bug reports and software changes.

3 Quality Assurance Indicator

We give, in this section, a summary of the definition of the *Quality Assurance Indicator* (Qi) metric. For more details see [22, 23]. The Qi metric is based on the concept of *Control Call Graphs* (CCG), which are a reduced form of traditional *Control Flow Graphs* (CFG). A CCG is a CFG from which the nodes representing instructions (or basic blocks of sequential instructions) not containing a call to a method are removed. The Qi metric is normalized and gives values in the interval [0, 1]. A low value of the Qi of a class means that the class is a high-risk class and a high value of the Qi of a class indicates that the class is a low-risk class.

3.1 Quality Assurance Indicator

The Qi of a method M_i is defined as a kind of estimation of the probability that the control flow will go through the method without any failure. The Qi of a method M_i is based on its intrinsic characteristics (*cyclomatic complexity, unit testing coverage*), as

well as on the Qi of the methods invoked by the method M_i. We assume that the quality of a method, particularly in terms of reliability, depends also on the quality of the methods it collaborates with to perform its task. In OO software systems, objects collaborate to achieve their respective responsibilities. A method of poor quality can have (directly or indirectly) a negative impact on the methods that use it. There is here a kind of propagation, depending on the distribution of the control flow in a system, which needs to be captured. It is not obvious, particularly in the case of large and complex OO software systems, to identify intuitively this type of interferences between classes (which is not captured by traditional OO metrics). The Qi of a method M_i is given by:

$$Qi_{M_i} = Qi^{*}_{M_i} \cdot \sum_{j=1}^{n_i} \left[P\left(C_j^i\right) \cdot \prod_{M \in \sigma_j} Qi_M \right] \quad (1)$$

with:

Qi_{M_i}: QA indicator of method M_i,

$Qi^{*}_{M_i}$: intrinsic QA indicator of method M_i,

C_j^i: j^{th} path of method M_i,

$P\left(C_j^i\right)$: probability of execution of path C_j^i of method M_i,

Qi_M: QA indicator of method M included in the path C_j^i,

n_i: number of linear paths of the CCG of method M_i,

and σ_j: set of the methods invoked in the path C_j^i.

By applying the previous formula (1) to each method we obtain a system of N (number of methods in the program) equations. The obtained system is not linear and is composed of several multivariate polynomials. We use an iterative method (method of successive approximations) to solve it. The system is, in fact, reduced to a fixed point problem. Furthermore, we define the Qi of a class C (noted Qi_C) as the product of the Qi of its methods:

$$Qi_C = \prod_{M \in \delta} Qi_M \quad (2)$$

where δ is the set of methods of the class C. The calculation of the Qi metric is entirely automated by a tool (prototype) that we developed for Java programs.

3.2 Assigning Probabilities

The CCG of a method can be seen as a set of paths that the control flow can pass through (depending on the states of the conditions in the control structures). To capture this probabilistic characteristic of the control flow, we assign a probability to each path C of a control call graph as follows:

$$P(C) = \prod_{A \in \theta} P(A) \quad (3)$$

where θ is the set of directed arcs composing the path C and P(A) the probability of an arc to be crossed when exiting a control structure.

Table 1. Assignment rules of the probabilities

Nodes	Probability Assignment Rule
(if, else)	0.5 for the exiting arc « condition = true » 0.5 for the exiting arc « condition=false »
while	0.75 for the exiting arc « condition = true » 0.25 for the exiting arc « condition = false »
(do, while)	1 for the arc: (the internal instructions are executed at least once)
(switch,case)	1/n for each arc of the n cases.
(?, :)	0.5 for the exiting arc « condition = true » 0.5 for the exiting arc « condition = false »
for	0.75 for entering the loop 0.25 for skipping the loop
(try, catch)	0.75 for the arc of the « try » bloc 0.25 for the arc of the « catch » bloc
Polymorphism	1/n for each of the eventual n calls.

To facilitate our experiments, we assigned probabilities to the different control structures of a (Java) program according to the rules given in Table 1. These values are assigned automatically during the static analysis of the source code of a program when generating the Qi models. These values can be adapted according to the nature of the applications (for example). As an alternative way, the probability values may also be assigned (adapted) by programmers during the development (in an iterative way, knowing the code) or obtained by dynamic analysis. Dynamic analysis is out of the scope of this paper.

3.3 Intrinsic Quality Assurance Indicator

The *Intrinsic Quality Assurance Indicator* of a method M_i, noted Qi^*_{Mi}, is given by:

$$Qi^*_{M_i} = (1 - F_i) \tag{4}$$

with: $F_i = \frac{(1-tc_i)cc_i}{cc_{max}}$

where:

CC_i : cyclomatic complexity of method M_i,

$cc_{max} = \max_{1 \le i \le N}(cc_i)$,

tc_i: unit testing coverage of the method M_i, $tc_i \in [0,1]$.

Many studies provided empirical evidence that there is a significant relationship between cyclomatic complexity and fault proneness (e.g., [13, 21, 32]). Testing (as one of the most important QA) activities will reduce the risk of a complex program and achieve its quality. Moreover, testing coverage provide objective measures on the effectiveness of a testing process. The testing coverage measures are (currently in our approach) affected by programmers based on the test suites they developed to test the classes of the program. The testing coverage measures can also be obtained automatically (using tools such as Together (`www.borland.com`) or CodePro (`developers.google.com`)) by analyzing the code of the test suites (JUnit suites for example) to determine which parts of the classes that are covered by the test suites and those that are not. This issue is out of the scope of this paper.

4 Empirical Study

We present, in this section, the empirical study we conducted in order to investigate if the Qi metric can be used to support the Lehman's laws. We focus on the following laws: *continuous change, increasing complexity, self-regulation, continuing growth* and *declining quality*. We used historical data collected from successive released versions of two open source Java software (Table 2).

Table 2. Some statistics on the used systems

Systems	Time frame (years)	Releases /Captures	First release computed			Last release computed		
			Version	Date	Size (SLOC)	Version	Date	Size (SLOC)
Eclipse PDE.UI	4.25	52 (monthly)	-	2002-08-29	9 519	-	2006-11-28	79 548
Apache Tomcat	7.2	31 (official)	5.5.0	-	126 927	5.5.35	-	170 998

4.1 The Case Studies

The first system we selected is an Eclipse component (PDE.UI). We used captures taken at regular intervals (monthly, more than four years). The second system we selected is a relatively large system (Tomcat). This system is an open source web server developed by the Apache Software Foundation. We analyzed its 5.5 branch, launched in August 2004. The version 5.5.35, the latest to date, was launched on November 2011. For this system, we used the official releases as time captures. The first version of PDE.UI includes 121 classes (more than 9 500 lines of code) and the last one includes 670 classes (more than 79 000 lines of code). The first version of Tomcat includes 837 classes (more than 120 000 lines of code) and the last one includes 1 108 classes (more than 170 000 lines of code).

Data Gathering. We collected two types of data from the subject systems for all the steps of our study: source code historical data and Qi data. In addition to these data, we also collected other (specific) data related to each law. These data will be presented in the corresponding sections.

System History Data. We used CVS (Concurrent Versions System) to collect historical data about PDE.UI. CVS is a client-server software that allows keeping track of a software system evolution. We connected to the CVS repertory of Eclipse. We based our analysis on the latest available version on the first day of each month. A period of more than four years (fifty two versions) is covered. For Apache Tomcat, we retrieved the official releases on the 5.5.x branch from the official website (archive.apache.org). A period of more than seven years (thirty one versions) is covered for this system.

Qi Data. We used the tool we developed to collect the Qi data. We computed the Qi value for each class of each released version of a subject system. We computed the Qi values at the micro level (classes) as well as at the macro level (system). Moreover, for our experiments, since we did not have any data on the test suites used for testing the subject systems and knowing that the main purpose of this study is to investigate if the Qi metric can be used to support Lehman's laws of software evolution, the test ing coverage (tc_i, Section 3.3) is set to 0.75 for all methods.

4.2 Lehman's Laws of Software Evolution

We consider in our study five of the eight Lehman's laws of software evolution: *continuing change, increasing complexity, self-regulation, continuing growth* and *declining quality*. We have not investigated the remaining two laws (conservation of organizational stability and conservation of familiarity) because we had no data to analyze them. For each of the considered laws, we investigate its applicability on our test applications. In addition, we describe the procedure we used, the specific data we collected (using different metrics to capture some attributes of the systems analyzed) in addition to the data described previously and the observations on whether the law is confirmed or infirmed. Our objective is to investigate if the Qi metric, as a synthetic metric, can be used to support the applicability of Lehman's laws of software evolution.

Continuing Change. An evolving system undergoes many changes over time. The Lehman's first law (continuing change) states that a software system must continually adapt to its environment, otherwise it becomes progressively less useful. Many studies on software evolution have addressed and validated this law for the open-source software systems studied [1, 9, 10, 33]. In our study, we consider change from the perspective of the cumulative number of added/removed classes. Figure 1 shows the curves of the cumulative number of changes for the two systems. The figure clearly shows that the two systems continue to change over time. Using linear regression, we calculated the slopes associated with these data sets. A similar approach has been used by Xie et al. [1]. From Figure 1, it can be seen that the slopes of the curves are positive for both systems (obviously since it is a sum of positive elements). Our focus is therefore on the strength of the slope of these curves. Results show that the changes take place periodically over time. However, changes in smaller amounts are held on a continuous basis and this throughout the period of evolution of the two systems. With these observations we can conclude that continuing change is confirmed for our two software systems.

We extended our analysis by focusing on variations in the values of some size metrics. We used several size indicators. We used the *number of lines of code in a system* (SLOC), the *total number of classes in a system* (SNOC) and the *average number of lines of code in a class* (LOC). We used the Borland Together tool (www.borland.com) to collect data on these metrics on the two subject systems. Figures 2 and 3 show, respectively for PDE.UI and Tomcat, variations in the used size related metrics. We can observe that the variations are quite dispersed for both systems over time. Therefore, changes in the case of the two systems are continuous. This confirms once again that continuous change is observable for the systems analyzed. Moreover, Figure 4 shows the variations in the Qi metric along the evolution of the two systems. The analysis of these curves clearly reveals continuing presence of changes in the case of both systems. Variations in the values of Qi are, in fact, continuous. With these observations, we can conclude that the Qi metric confirms also the first law for the systems analyzed.

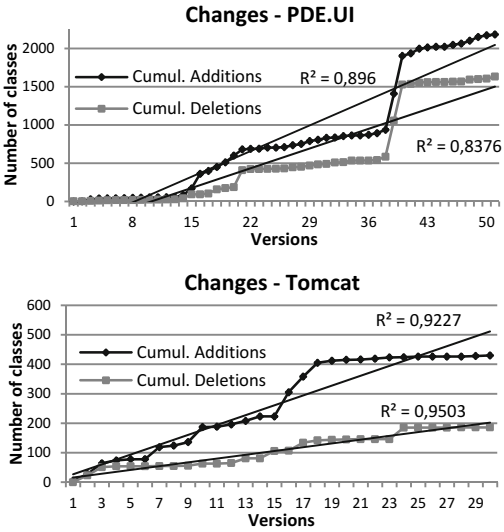


Fig. 1. Evolution of cumulative changes for PDE.UI and Tomcat

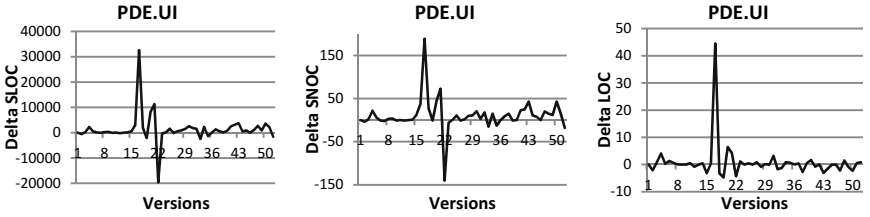


Fig. 2. Evolution of size metrics variations for PDE.UI

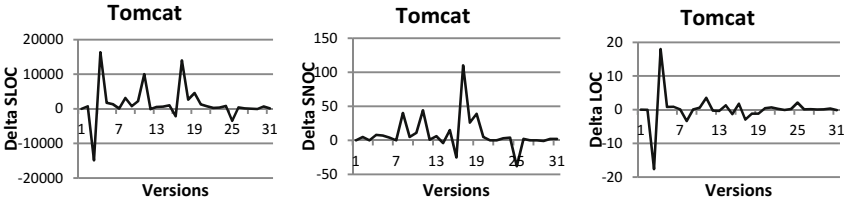


Fig. 3. Evolution of size metrics variations for Tomcat

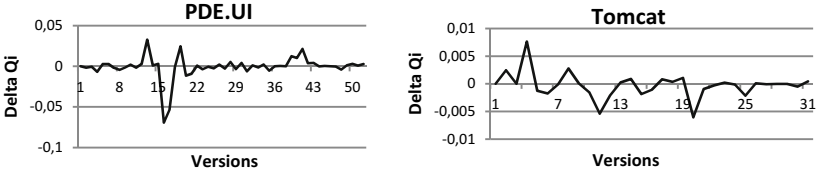


Fig. 4. Evolution of Qi variations for PDE.UI and Tomcat

Increasing Complexity. The second law states that as a software system evolves its complexity increases unless work is done to reduce or stabilize it. Studies that addressed Lehman's second law in the case of open-source development have confirmed this law for the systems analyzed [1, 9]. In our study, we used three well-known software metrics (RFC, WMC and CBO) [34] to capture complexity and coupling along the evolution of the systems studied. The CBO (*Coupling Between Objects*) metric counts for a class the number of other classes to which it is coupled (and vice versa). The RFC (*Response For Class*) metric for a class is defined as the set of methods that can be executed in response to a message received by an object of the class. The WMC (*Weighted Methods per Class*) metric gives the sum of complexities of the methods of a given class, where each method is weighted by its cyclomatic complexity. Here also, we used the Borland Together tool to collect data on these metrics on the two subject systems. We calculated the values of these metrics for all versions of the considered systems. Figures 5 and 6 show, respectively for PDE.UI and Tomcat, the curves of the metrics and the slope of the linear regression followed by each of these curves.

Overall, from the two figures, we can clearly observe that all the curves show an increasing trend. The three metrics have increased over time. With these observations, we can conclude that the second law is confirmed for the two systems. Moreover, Figure 7 shows the evolution of the Qi metric for both systems. From this figure, we can observe a decreasing trend of the Qi metric, which is opposite to the trend of the curves corresponding to the coupling and complexity metrics. The decreasing in the values of the Qi metric confirms also the second law for the two systems. We also calculated the correlation coefficients between the Qi metric and the coupling and complexity metrics. We used the Spearman's correlation coefficient in our study. This technique is widely used for measuring the degree of relationship between two variables. Correlation coefficients will take a value between -1 and +1. We applied the typical significance threshold ($\alpha = 0.05$) to decide whether the correlations were significant. We analyzed the correlations between the Qi metric and the coupling and complexity metrics at both micro and macro levels.

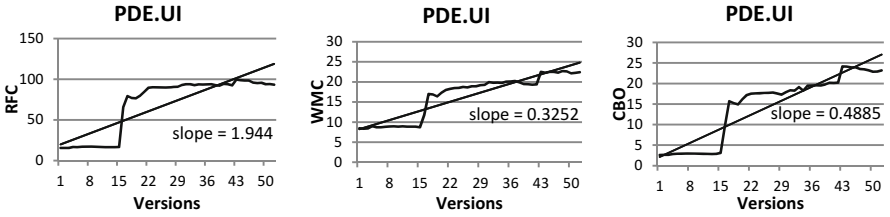


Fig. 5. Evolution of coupling and complexity metrics for PDE.UI

For the macro level, we used the average values of the metrics for each version of the subject systems. We give in what follows (Table 3), for space constraints, only the correlation values at the macro level. From Table 3, it can be seen that all correlations are significant (in boldface). The correlation values are moderate for PDE.UI but relatively high (especially between the Qi metric and RFC and CBO metrics) for Tomcat, the larger of the two systems. Moreover, the correlations are negative.

This confirms the trends observed from the different previous curves. Therefore, we can conclude that the increasing complexity of the two systems is once again confirmed and supported by the Qi metric.

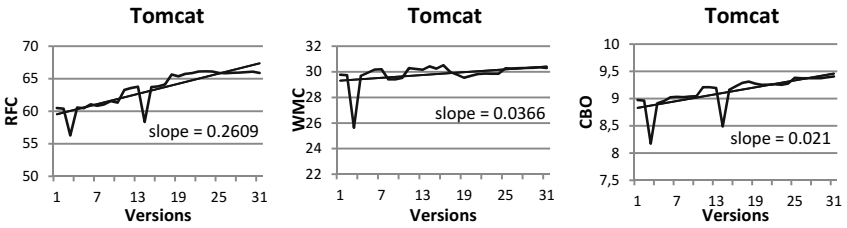


Fig. 6. Evolution of coupling and complexity metrics for Tomcat

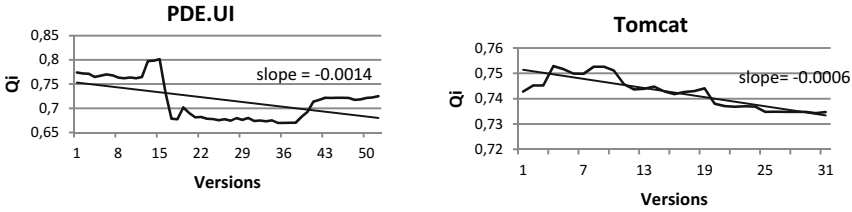


Fig. 7. Evolution of the Qi metric for PDE.UI and Tomcat

Table 3. Correlations between Qi and complexity metrics for PDE.UI and Tomcat

	RFC vs. Qi	WMC vs. Qi	CBO vs. Qi
PDE.UI	**-0.516**	**-0.539**	**-0.475**
Apache Tomcat	**-0.830**	**-0.566**	**-0.854**

Self-regulation of Large Systems. The third law states that large systems have their own internal dynamic that makes that their size adjust itself through time. G. Xie et al. [1] presented a simple way to observe self-regulation of the size of systems by analyzing variations in classes searching for fluctuations (alternating from positive to negative). We used a similar approach for observing adjustments in the values of the size attributes. We used variations (deltas) of size metrics (SLOC, SNOC and LOC) presented in section – Continuing Change. We focused on the sign of these variations. Figures 2 and 3 effectively allow us to see the positive and negative deltas. The positive deltas happen far more often than negative ones. However, the negative deltas are observable at some dispersed moments throughout the evolution of the two systems analyzed. By using Qi, similar trends can be observed. We can see in figure 4 that both variations are observable, except that this time the negative deltas are more frequent than the positive deltas. Here also, both variations are distributed regularly throughout the evolution of the two systems. This means that there is a self-regulation in the Qi values for the two systems. With these results (size metrics and Qi), we can confirm the law of self-regulation for the two selected systems.

Continuing Growth. This law states that the size of a software system increases continuously along its evolution to accommodate changes and satisfy an increasing set of requirements. Many studies that addressed this law were able to confirm it for the open-source software systems studied [1, 9, 30, 33]. These studies used different metrics for measuring system size and growth. We used the number of classes as an indicator of size. This attribute has been used by Lee et al. [9] to confirm continuing growth. Figure 8 shows the evolution of the number of classes for both systems. Each point in the graph corresponds to a release. We can clearly see that the size of both systems, measured by the number of classes, increased over time, which confirms the law for both systems. We extended our analysis by using other size indicators. In addition to the size indicators SLOC, SNOC and LOC used in section – Continuing Change, we used the *number of operations per class* (NOO). The curves of these indicators, which we do not give in this paper for space constraints, follow exactly the same trend as the curve of Figure 8. All these indicators increased over time for both systems. Moreover, we can also consider the additions/deletions of classes presented previously in section – Continuing Change (Figure 1), in which the curve of additions is above the deletions. The cumulative number of additions is thus clearly higher (growths faster) than the cumulative number of deletions. With these observations we can confirm the law of continuing growth for both systems.

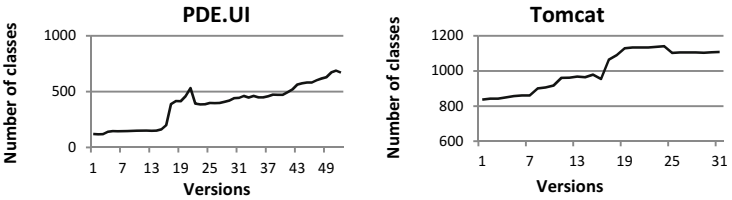


Fig. 8. Evolution of the number of classes of PDE.UI and Tomcat

Let us now analyze the evolution of the Qi metric. From Figure 7, we can observe that the Qi metric follows an opposite trend compared to the size indicators, which is not a surprising finding. In fact, a significant increase in the size of a software system is generally followed by an increase in its complexity, which leads to a decrease in the Qi values over time, as explained in section – Increasing Complexity. In order to validate these observations, we analyzed the correlation values between the Qi metric and the size indicators. We used here also the Spearman's correlation coefficient under the same conditions as in section – Increasing Complexity. Table 4 gives the obtained results. As it can be seen, the correlations are all significant (in boldface) and negatives. This confirms that the Qi metric supports also the law of continuing growth for both systems.

Table 4. Correlations between Qi and size metrics for PDE.UI and Tomcat

	SNOC vs. Qi	SLOC vs. Qi	LOC vs. Qi	NOO vs. Qi
PDE.UI	**-0.433**	**-0.442**	**-0.891**	**-0.561**
Tomcat	**-0.736**	**-0.790**	**-0.397**	**-0.697**

Declining Quality. This law states that the quality of a software system decreases over time, unless work is done to improve it. Among the studies that have investigated this law [1, 9], none was able to confirm it for the test systems they used. The study conducted by Lee et al. [9], in particular, has observed an overall improvement of software quality. In our study, we address this law from different perspectives. In a first step, we considered software quality from an external perspective. In order to analyze how software quality changes as software evolves, we used the number of reported defects as a quality indicator. We also investigated for patterns of evolution (according to Zhang et al. [7]). A higher occurrence of faults is considered as a sign of poor quality. We used, in the case of the two systems, Bugzilla (defect tracking system) reports on resolved/closed faults to withdraw information about faults. Figure 9 shows the evolution of the number of faults identified at each iteration of the two systems. Zhang et al. [7] studied the evolution of faults of PDE.UI based on recurring patterns. The authors found that PDE.UI follows a roller coaster pattern, which means that its quality is not under control. Amplitudes are, in fact, at their strongest in the center of the evolution period studied (iterations 21 and 33). With the exception of these two periods, the quality seems to be much more under control. The results we obtained for PDE.UI confirm the conclusions of Zhang et al. [7]. Tomcat has not been studied by Zhang et al. [7] However, based on the patterns they identified, we can say that this system also follows a roller coaster pattern due to the many peaks visible in the curve. The quality of Tomcat, however, seems to be much more under control towards the end of the analyzed period with a stabilization of the number of faults. Such observations therefore lead us to conclude in a deterioration of the quality for the two systems under study. In addition, we considered in a second step software quality from an internal point of view. From the curves given in Figure 5 and Figure 6 (increasing complexity), we can clearly observe that the complexity of the two systems increases, according to the three metrics (RFC, WMC and CBO). These observations confirm therefore that the quality of the analyzed systems from an internal point of view decreases. In fact, an increasing complexity of a software system often symbolizes a decrease in its quality. So, when considering both internal and external quality metrics, we can conclude that the law of declining quality is confirmed for the two systems.

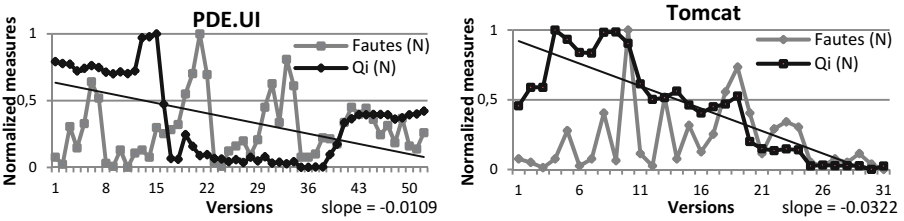


Fig. 9. Evolution of the number of faults superimposed with Qi for the two systems

Let us now analyze the evolution of the Qi metric. Figure 9 shows the curve of the Qi metric, superimposed with the number of faults reported along the evolution of the two systems. The linear regression of the Qi curve is also given in the figure. The values on the y-axis are normalized between 0 and 1 (min-max). From Figure 9

and previous results, we can make several observations. PDE.UI shows a negative trend for the Qi metric, mainly because of the steep decrease between iterations 15 and 17. To this period corresponds, in fact, a significant increase in complexity (see Figure 5) and a significant growth of the system (see Figure 8). Immediately after, we can observe a long plateau where the Qi values are at their lowest, ranging between iterations 17 and 38. Throughout this period, the number of faults is relatively high and the complexity metrics remain relatively stable. Two large peaks appear in the curve of faults around iterations 20 and 32. After this long period, the Qi values rebounded. Overall, for PDE.UI, we can conclude that the Qi metric thus indicates the variations (especially for the steep drop) of quality over time. Tomcat presents for the Qi metric a regular decreasing trend, during which faults appear as peaks in regular intervals. In addition, we already know from previous results that there is an increasing complexity for this system, according to the evolution of the complexity metrics (CBO, RFC and WMC). From these observations, we can conclude that the Qi metric captures the declining quality of the two systems. We can therefore say that the Qi metric supports the declining quality law for the studied systems.

Threats to Validity. The study presented in this paper should be replicated using many other software systems in order to draw more general conclusions about the ability of the Qi metric to support the Lehman's laws of software evolution. In fact, there are a number of limitations that may affect the results of the study or limit their interpretation and generalization. The achieved results are based on the data set we collected from only two open source software systems written in Java. Even if the collected data cover a period of several years: 4 years for the first system (fifty-two versions) and 7 years for the second one (thirty-one versions), we do not claim that our results can be generalized to all systems, or software systems written in other languages. The findings in this paper should be viewed as exploratory and indicative rather than conclusive. Moreover, the study has been performed on open source software systems. It would be interesting to replicate the study on industrial systems. It is also possible that facts such as the development style used by the developers for developing (and maintaining) the code of the subject systems (or other related factors) may affect the results or produce different results for specific applications. In addition, the study is based implicitly on the assumption that the used software metrics (LOC, RFC, WMC, CBO, etc.) actually capture the intended characteristics. We deliberately used multiple metrics for each law to reduce this threat. Also, the study is based on the data we collected on the evolution of the studied systems, in particular the defect information, that we suppose reliable.

5 Conclusions and Future Work

In this paper, we analyzed the evolution of two open-source Java software systems. We wanted to investigate if the Qi (*Quality Assurance Indicator*) metric, a metric that we proposed in a previous work, can be used to support the applicability of Lehman's laws of software evolution. We focused in this study on five of the Lehman's laws of software evolution: continuing change, increasing complexity, self-regulation, continuing growth and declining quality. We addressed software evolution from both internal and external perspectives. We performed an empirical analysis using historical

data collected from the successive released versions of the two systems. The collected data cover a period of more than four years for the first system (fifty-two versions in total) and a period of more than seven years for the second one (thirty-one versions in total). Empirical results provide evidence that the considered Lehman's laws are supported by the collected data and the Qi metric.

The advantage that brings, in our opinion, the use of the Qi metric (as a synthetic metric) in the case of evolving systems, is that it can be used to guide quality assurance activities through evolution. Indeed, the Qi metric can be used for identifying, in a relative way as software evolves, critical parts that require more quality assurance (as testing) effort to ensure software quality. The achieved results are, however, based on the data set we collected from only two open source software systems. The findings in this paper should be viewed as exploratory and indicative rather than conclusive. They show, at least, that the Qi metric, as a synthetic metric, offers a promising potential for capturing (reflecting) various aspects related to software evolution. Further investigations are, however, needed to draw more general conclusions. In addition, we plan to explore the potential of the Qi metric to support predictive analysis.

Acknowledgements. This project was financially supported by NSERC (National Sciences and Engineering Research Council of Canada) and FRQNT (Fonds de Recherche du Québec – Nature et Technologies) grants.

References

1. Xie, G., Chen, J., Neamtiu, I.: Towards a better understanding of software evolution: An empirical study on open source software. In: ICSM 2009, pp. 51–60 (2009)
2. Sommerville, I.: Software engineering, 9th edn. Addison Wesley (2010)
3. Parnas, P.L.: Software aging. In: Proceedings of the 16th ICSE, pp. 279–287 (1994)
4. Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski, W.M.: Metrics and laws of software evolution – The nineties view. In: Proceedings of the Fourth International Software Metrics Symposium, pp. 20–32 (1997)
5. Lehman, M.M.: Laws of software evolution revisited. In: Montangero, C. (ed.) EWSPT 1996. LNCS, vol. 1149, pp. 108–124. Springer, Heidelberg (1996)
6. van Gurp, J., Bosch, J.: Design erosion: Problems & causes. Journal of Systems and Software 61(2), 105–119 (2002)
7. Zhang, H., Kim, S.: Monitoring software quality evolution for defects. IEEE Software 27(4), 58–64 (2010)
8. Lehman, M.M., Belady, L.A.: Program evolution: Processes of software change. Academic Press (1985)
9. Lee, Y., Yang, J., Chang, K.H.: Metrics and evolution in open source software. In: Proceedings of the 7th QSIC (2007)
10. Mens, T., Demeyer, S.: Future trends in software evolution metrics. In: Proceedings of the 4th IWPSE, pp. 83–86 (2001)
11. Henderson-Sellers, B.: Object-oriented metrics – Measures of complexity. Prentice Hall, New Jersey (1996)
12. Badri, M., Touré, F.: Empirical analysis for investigating the effect of control flow dependencies on testability of classes. In: 23rd International Conference on Software Engineering and Knowledge Engineering (2011)

13. Basili, V., Briand, L., Melo, W.L.: A validation of object oriented design metrics as quality indicators. IEEE Transactions on Software Engineering 22(10) (1996)
14. Briand, L.C., Wüst, J., Daly, J.W., Porter, D.V.: Exploring the relationships between design measures and software quality in object-oriented systems. Journal of Systems and Software, 245–273 (2000)
15. Dagpinar, M., Jahnke, J.H.: Predicting maintainability with object-oriented metrics – An empirical comparison. In: Proceedings of the 10th Working Conference on Reverse Engineering, pp. 155–164 (2003)
16. Eski, S., Buzluca, F.: An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes. In: 2011 IEEE 4th Int. Conference on Software Testing, V&V Workshops, pp. 566–571 (2011)
17. Fenton, N.E., Pfleeger, S.L.: Software metrics: A rigorous & practical approach, 2nd edn. PWS Publishing Company (1997)
18. Murgia, A., Concas, G., Pinna, S., Tonelli, R., Turnu, I.: Empirical study of software quality evolution in open source projects using agile practices. In: CoRR, Vol. abs/0905.3287 (2009)
19. Singh, Y., Kaur, A., Malhotra, R.: Empirical validation of object-oriented metrics for predicting fault proneness models. Software Quality Journal 18(1), 3–35 (2010)
20. Subramanyan, R., Krishnan, M.S.: Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. IEEE Transactions on Software Engineering 29(4), 297–310 (2003)
21. Zhou, Y., Leung, H.: Empirical analysis of object-oriented design metrics for predicting high and low severity faults. IEEE Transactions on Software Engineering 32(10), 771–789 (2006)
22. Badri, M., Badri, L., Touré, F.: Empirical analysis of object-oriented design metrics: Towards a new metric using control flow paths and probabilities. Journal of Object Technology 8(6), 123–142 (2009)
23. Badri, M., Touré, F.: Evaluating the effect of control flow on the unit testing effort of classes: An empirical analysis. Advances in Software Engineering Journal (2012)
24. Badri, M., Drouin, N., Touré, F.: On Understanding Software Quality Evolution from a Defect Perspective: A Case Study on an Open Source Software System. In: Proceedings of the IEEE International Conference on Computer Systems and Industrial Informatics, Sharjah, UAE, December 18-20 (2012)
25. Drouin, N., Badri, M., Touré, F.: Metrics and Software Quality Evolution: A Case Study on Open Source Software. In: Proceedings of the 5th International Conference on Computer Science and Information Technology, Hong Kong, December 29-30 (2012)
26. Nagappan, N., Ball, T., Zeller, A.: Mining metrics to predict component failures. In: Proceedings of the 28th International Conference on Software Engineering (ICSE 2006), pp. 452–461. ACM (2006)
27. Ambu, W., Concas, G., Marchesi, M., Pinna, S.: Studying the evolution of quality metrics in an agile/Distributed project. In: Abrahamsson, P., Marchesi, M., Succi, G. (eds.) XP 2006. LNCS, vol. 4044, pp. 85–93. Springer, Heidelberg (2006)
28. Lehman, M.M.: On understanding laws, evolution, and conservation in the large-program life cycle. Journal of Systems and Software 1(3), 213–221 (1980)
29. Jermakovics, A., Scotto, M., Succi, G.: Visual identification of software evolution patterns. In: 9th International Workshop on Principles of Software Evolution (IWPSE 2007): in Conjunction with the 6th ESEC/FSE Joint Meeting, pp. 27–30 (2007)
30. Mens, T., Fernandez-Ramil, J., Degrandsart, S.: The Evolution of Eclipse. In: IEEE ICSM, pp. 386–395 (2008)

31. Yu, L., Ramaswamy, S., Nail, A.: Using bug reports as a software quality measure. In: Proceedings of the 16th ICIQ (2011)
32. Aggarwal, K.K., Singh, Y., Kaur, A., Lalhotra, R.: Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: A replicated case study. Software Process: Improvement and Practice 16(1) (2009)
33. Fernandez-Ramil, J., Lozano, A., Wermelinger, M., Capiluppi, A.: Empirical studies of Open-Source Evolution. In: Mens, T., Demeyer, S. (eds.) Software Evolution, pp. 263–288. Springer, Berlin (2008)
34. Chidamber, S.R., Kemerer, C.F.: A metric suite for object-oriented design. IEEE Transactions on Software Engineering 20(6), 476–493 (1994)

Automatic Extraction of Behavioral Models from Distributed Systems and Services

Ioana Şora and Doru-Thom Popovici

Department of Computer and Software Engineering,
Politehnica University of Timisoara, Romania

Abstract. Many techniques used for discovering faults and vulnerabilities in distributed systems and services require as inputs formal behavioral models of the systems under validation. Such models are traditionally written by hand, according to the specifications which are known, leading to a gap between the real systems which have to be validated and their abstract models.

A method to bridge this gap is to develop tools that automatically extract the models directly from the implementations of distributed systems and services. We propose here a general model extraction solution, applicable to several service technologies. At the core of our solution we develop a method for transforming the control flow graph of an abstract communicating system into its corresponding behavioral model represented as an Extended Finite State Machine. We then illustrate our method for extracting models from services implemented using different concrete technologies such as Java RMI, Web services and HTTP Web applications and servlets.

Keywords: Reverse Engineering, Behavioral Model, EFSM, Distributed Computing, Service Computing.

1 Introduction

Important research efforts aim at improving security in the Internet of Services by developing a new generation of security analyzers for service deployment, provision and consumption [16]. The techniques used for discovering faults and vulnerabilities comprise model checking [3] or model based testing [5]. All these techniques take as input a model of the system under validation and the expected security goals, expressed in a specific description formalism. Usually the models are hand written by the security analyst, based on the service specifications. This approach has been successfully used in the discovery of protocol errors, of logical errors which are present in the known models of systems, or the discovery of errors due to the interaction of known systems.

One of the factors which can promote the use of these validation techniques is given by how easy it is to produce the models which are required as inputs by the various validation tools. Also, these models should reflect with accuracy the real system. It results that relying on hand-written models is not always a suitable approach: it is the case of service implementers, who must make sure that the model reflects the actual implementation, and it is the case of service consumers who use black-box services from third party providers and need a reliable model of it.

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 190–202, 2013.

In this work, we focus on getting service models at service implementation and deployment time. Service developers could benefit more from the large variety of tools for security analysis and validation, such as the SPaCIoS tool [16], if they had model-extractor tools able to extract behavioral models from service implementations. Currently they have to manually write such models using the Aslan++ specification language [12]. Our current work [14] addresses this issue of extracting behavioral models from service implementations, by applying specific white box techniques based on the analysis of their control flow graph.

The difficulty in analyzing the code of real service implementations comes from the complexity of the code, which is usually written using different technologies frameworks and APIs. It is not possible to obtain models of a reasonable high abstraction level without taking into account the speciffics of each API which is used by providing special abstractions for them. Doing so, the disadvantage is that it leads to dedicated model extraction tools, each designed to handle systems or services implemented in a given technology.

Our approach handles the different technologies by identifying how they map onto a set of general abstract communication operations. Then, the problem of extracting models of systems implemented using different technologies and frameworks is split into two distinct subproblems: first, the problem of extracting the model of a system which uses only a set of abstract communication operations, and second, the problem of mapping these abstract communication operations onto the operations of different frameworks and APIs for distributed systems and services development. According to this, the model extractor tool comprises a stable, general model extractor core and a set of technology-dependent preprocessing frontends, like depicted in Figure 1.

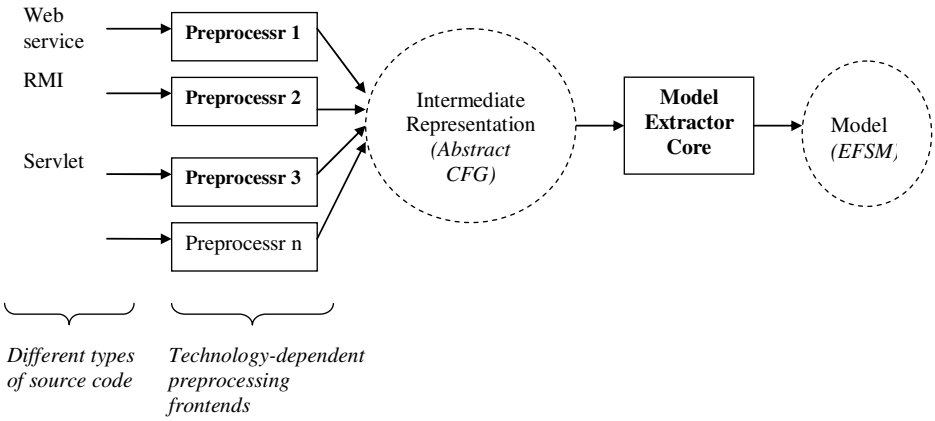


Fig. 1. The two steps of the model extraction approach

The remainder of this article is organized as follows. Section 2 presents background information about representing behavioral models as extended finite state machines. Section 3 presents the specific ways of mapping complex constructions of different frameworks and APIs for the construction of distributed systems and services into

abstract message communication operations. We then define our method of model extraction in abstract, technology-independent terms in Section 4. We discuss aspects related to our approach in Section 5.

2 Extended Finite State Machines Used for Behavioral Modeling

In this work we use a form of Extended Finite State Machines (EFSM) for representing behavioral models. Our EFSMs are Mealy machine models which are specifically tailored for white-box modeling of I/O based systems. Further, such models can be translated into Aslan++ [12] or into another language for modeling of distributed systems and services.

We consider as I/O the messages exchanged by the system with its environment. Each message is characterized by a message type and a set of message parameters which may have different values. The input alphabet of the EFSM is the set RM of all message types rm, which may be received by the system. The output alphabet of the EFSM is the set SM of all message types sm, which may be sent by the system. For each message type m, $m \in RM$ or $m \in SM$, the set of parameter types $P(m)$ is known.

Since the model is extracted through white-box techniques, it may also contain and use without restrictions state variables $v \in V$ which are not directly observable from the exterior but they can be extracted from the code.

An EFSM model consists of S, the set of all states s, with only one state being the initial state s_0, a set T of all transitions t between states, and V the set of all state variables v.

A transition t is defined by six components: its origin state $s_i \in S$, its destination state $s_j \in S$, the received message $rm \in RM$, the guard predicate g, the action list al, the sent message $sm \in SM$.

A transition t between two states s_i and s_j occurs when a message rm is received and a guard condition predicate g is true. In this case, the list of actions associated with the transition al is executed and a message sm is sent.

```
If (ReceiveMsg (rm) and isTrue(g)) then
    doActions(al);
    SendMsg (sm);
```

It is possible that some of the following components of a transition are missing: rm, g, al, sm.

State variables and parameters may be scalar variables or sets.

A guard condition predicate g is a boolean expression. The operands of the guard predicate g on a transition fired by a received message rm with a set of parameters $P(rm)$ can be both state variables $v \in V$ and parameters of the received message $p \in P(rm)$. The operators can be boolean operators (and, or, not), relational operators, or set operators (contains).

A list of actions al is an ordered sequence of actions a_i. An action a_i on a transition fired by a received message rm with a set of parameters $P(rm)$, which sends a message sm with a set of parameters $P(sm)$ is an assignment. The left value of the assignment

is a state variable $v \in V$ or a parameter of the sent message $p \in P(sm)$. The right value of the assignment is an expression which can have as operands state variables $v \in V$, or parameters of the received message $p \in P(rm)$. Operators are boolean, relational and set operators (add to, remove from).

3 Modeling Services of Different Technologies

Our work aims at modeling distributed systems and services in form of EFSMs as presented in section 2. An application or service can be implemented in different ways using different technologies, but still be described by the same behavioral model. In the next subsection we introduce a system which will be used as a running example, together with the EFSM representing its behavioral model, while next subsections use different technologies to implement the same system. This helps identifying how the specific constructs of different APIs can be mapped into a set of abstract message sending operations and leads to defining the tasks that have to be performed by the technology-dependent frontends of the model extractor tool in order to produce an intermediate system representation as an abstract control flow graph.

3.1 A Running Example

We introduce the following Online Shop as a running example. The Online Shop acts as a server which may receive commands for ordering goods, paying for them, and requesting that the payed products are delivered.

We assume that the server receives and sends messages, by explicit messaging operations such $SendMessage$ and $ReceiveMessage$. The input alphabet (the set of received message types RM) comprises: `orderType`, `payType`, `deliveryType`, while the output alphabet (the set of sent mesage types SM) comprises `deliveryResp`. The received messages of all types take one parameter `name` which serves as the identifyer of orders, payments and deliveries. The functioning of the shop assumes that for a name, an order has to be placed first, then it can be payed and only after that it can be delivered. In order to keep track of the state of orders which have been submitted and payments which have been done, the model employs two state variables, `orders` and `payments`, which are sets of names. The Online Shop is modeled as an EFSM with two states, the initial state and the state corresponding to the server loop state. Initially, the sets `orders` and `payments` are initialized as empty sets. In the server loop state, the system may receive messages of the types `orderType`, `payType`, `deliveryType`. These determine transitions which go into the same server loop state, but the actions and mesages sent are different, according to the message received and a set of guard conditions.

Figure 2 presents the EFSM of the simple Shop server. In this figure we shortened for presentation purposes the names: the message types are denoted by `o`, `p`, `d`, and `dR` (for `orderType`, `payType`, `deliveryType`, and `deliveryResp`), the parameter `name` is denoted `n`, the state variables `orders` and `payments` are named `os` and `ps`.

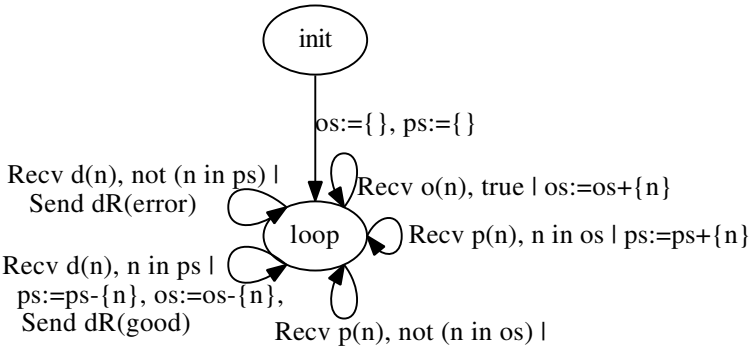


Fig. 2. Example: EFSM model of simple Shop server

3.2 Technologies Used for Implementation of Distributed Systems and Services

In practice, such an Online Shop server corresponding to the above model can be implemented using a large variety of different technologies, frameworks and APIs for distributed systems and services. These help the application developer to cope with the complexity of such systems, but performing code analysis becomes more difficult for the following two reasons:

- Instead of explicit $SendMessage$ and $ReceiveMessage$ instructions, frameworks offer complex APIs to describe the interactions of a server.

  The first step towards applying our model extraction method is to identify for each API the constructions which are equivalent with sending and receiving messages and define abstractions for them.
- Frameworks also provide infrastructure support for the execution of developed applications. Most often, by analyzing only the application code written by the application developer one cannot obtain the whole control flow graph (CFG) of the real system. For example, in all frameworks the application developer does not explicitly provide the server loop, which is something that is added by default through the framework.

  The particularities of each framework have to be known and the partial CFG or CFGs extracted from the application code must be completed or combined in order to obtain the complete CFG.

These issues (identifying and abstracting send/receive message operations, completing the partial CFG from application code) have to be solved by technology specific preprocessing frontends before the generic model construction method presented in 4.2 may proceed.

Our current work considers modeling servers which are implemented in Java and according to a set of specific technologies. The limitation to analyzing only Java code is a temporary one, due to the fact that we need specific support for static code analysis for each new programming language. The basics of our method are set by building blocks for static code analysis such as: call graph construction, inter-procedural control flow

graph construction, and data flow analysis. For implementation we focused on systems implemented in the Java programming language because we can rely on these building blocks offered by the Watson Libraries for Analysis (WALA) [8].

We categorize these technologies as being with or without explicit interfaces. Technologies such as WSDL Web Services, Java RMI, and CORBA, make the interfaces of the services explicit, either as language interfaces or as interfaces described in a special interface description language. Other technologies such as Servlets or JSP do not make the interfaces explicit. The following subsections detail how the explicit constructions of these technologies are mapped into abstract $SendMessage$ and $ReceiveMessage$ operations and how the corresponding preprocessing frontends produce the abstract control flow graph.

3.3 Preprocessing Frontend for Interface-Explicit Technologies

In the case of Java RMI, but also in case of other interface-explicit technologies such as WSDL Java Web services, a server is a special kind of object, implementing the methods described in an explicit interface. The interface description contains the list of possible operations, with their full signature (method name, number and types of parameters, return type). Clients can interact with a server invoking these methods. These are the entrypoints of the server application.

The Online Shop can be implemented as a RMI server, by first defining its interface as a Java interface which extends the `rmi.Remote`interface and then defining a Java class which implements this interface:

```
public class ShopImpl
                extends UnicastRemoteObject
                implements  ShopInterface {

  private Set<String> orders = new HashSet<String>();
  private Set<String> payments = new HashSet<String>();

  public synchronized void order(String name)
                          throws RemoteException {
      orders.add(name);
  }

  public synchronized void pay(String name)
                        throws RemoteException {
      if (orders.contains(name)) {
        orders.remove(name);
        payments.add(name);
        }
     }
  public synchronized String get(String name)
                        throws RemoteException {
    if (payments.contains(name)) {
       payments.remove(name);
       return new String("YourProduct");
```

```
      }
    else return new String("NotPayed");
}
}
```

The entry points for a RMI application are those methods declared in an interface that extends the `rmi.Remote` interface. When analyzing an application that uses RMI, the preprocessing frontend looks for this kind of methods as entrypoints.

We can define the needed $SendMessage$ and $ReceiveMessage$ abstractions in RMI code in the following way: A RMI object receives a message when one of its remote methods is invoked. Thus the entrypoint of every remote method is modeled as an abstract $ReceiveMessage$ operation. A RMI object sends a message when returning from a remote method invocation or when raising an exception.

Names for message types are derived automatically from method names. The type of the sent message differs from the type of the received message corresponding to the method invocation (it is a return-methodname type of message). The parameters of the received message correspond to the arguments of the method. The parameters of the sent message correspond to the returned values or exceptions raised.

For example, a method with following signature:

```
String deliver(String name) {
   ... // some statements
}
```

will be abstracted to:

```
ReceiveMessage deliverType, name
... // some statements
SendMessage deliverResp, aString
```

By analyzing the RMI application code, the CFGs of each entrypoint method can be built. In order to get the whole CFG of the RMI server, all these partial CFGs have to be framed by a server loop and preceded by the initialization code. After these preprocessing are done, the core model construction algorithm can be applied on the adjusted CFG.

3.4 Preprocessing Frontend for Servlets and JSP

Web applications are dynamic extensions of web or application servers, which may generate interactive web pages with dynamic content in response to requests. In the Java EE platform, the web components which provide these dynamic extension capabilities are either Java servlets or Java Server Pages (JSP).

A servlet is a Java class that conforms to the Java Servlet API, which establishes the protocol by which it responds to HTTP requests, and generates dynamic web content as response. The popular JSP technology, which embeds Java code into HTML, relies on Servlets, as these are automatically generated by the application server from JSP pages. When analyzing JSP pages, we first explicitly call the JSP compiler in order to obtain the source code of their corresponding servlet classes.

In the code analysis, we identify Java Servlets as the classes that extend the `javax.servlet.HttpServlet` class. Their entrypoints are the methods: `doGet`, `doDelete`, `doHead`, `doOptions`, `doPost`, `doPut`, `doTrace`, `service`. The servlets generated from JSP are classes which extend `org.apache.jasper.runtime. HttpJspBase` and their entrypoints are methods `jspInit()` and `jspService()`.

When analyzing an application that uses servlets, the preprocessing frontend looks for this kind of methods as entrypoints. Similarly to the RMI preprocessor, the CFGs of each entrypoint can be built and in order to get the whole CFG all these partial CFGs have to be framed by a server loop and preceded by the initialization code.

All entrypoint methods have as parameters `HttpServletRequest` and `HttpServletResponse`, which correspond to the types of the messages which are sent and received.

The `HttpServletRequest` allows access to all incoming data. The class has methods for retrieving form (query) data, HTTP request headers, and client information. The `HttpServletResponse` specifies all outgoing information, such as HTTP status codes, response headers, cookies, and also has a method of retrieving a `PrintWriter` used to create the HTML document which is sent to the client.

What is different and more difficult in this case is abstracting the parameters of the SendMessage and ReceiveMessage statements. Message parameters cannot be identified directly in this case, since incoming and outgoing data are handled through a large number of specific methods on the request and response objects.

As an example, we consider below an excerpt of the Online Shop example, this time in an implementation with servlets:

```
public class Shop extends HttpServlet {
  // ... omitted parts
  protected void doGet(HttpServletRequest req,
                       HttpServletResponse resp)
    // ...  parts are omitted or simplified
    String op = req.getParameter("operation");
    if (op.equals("deliver"))   {
        String name=req.getParameter("name");
        if (payments.contains(name))
             delivResp=doService();
        else delivResp=error();
        Writer w=response.getWriter();
        w.write(delivResp);
        }
    else if (op.equals("pay"))
    // ...
```

Abstracting send and receive operations with parameters from the code of each entry point method is a complex task which must take into account every method that can be called on a `HttpServletRequest` or `HttpServletResponse` object.

The entrypoiny of the method corresponds to a $ReceiveMessage$ statement, receiving a message of type $HttpRequest$. The parameters of this received message may contain: a set of name - value pairs, corresponding to the `ParameterMap`, and a

set corresponding to the Session attributes. The parameters will be added to the ReceiveMessage statement only if they are used in the method body: the first parameter will be added only if the method body contains statements for retrieving the ParameterMap or specific parameters from the request object. The second parameter will be added only if there are statements retrieving a Session from the request object and getting values from there.

In our example, we have only calls of method `getParameter` on the `request` object, no `Session` object has been retrieved and used, thus the received abstract message is:

```
ReceiveMessage HttpRequest (
   ("operation", op), ("name", name))
```

Each path leading to an exit point of the method will end in a $SendMessage$ statement, sending a message of type $HttpResponse$. The parameters of this sent message are: all the variables which are written by the output Writer along this path, and session attributes if they have been retrieved and handled in the method body.

In our example, following $SendMessage$ statements are abstracted on the different paths:

```
SendMessage HttpResponse (delivResp)
SendMessage HttpResponse ("Order finished")
SendMessage HttpResponse ("Pay finished")
```

4 From (abstract) Control Flow Graph to Extended Finite State Machine

4.1 Preliminary Assumptions

We present the principles of our model inference algorithm starting from the following assumptions:

- The system is described by a complete, inter-procedural Control Flow Graph (CFG).
- There are explicit statements, corresponding to a node in the CFG, for receiving and sending messages of a specified message type and having message parameters.

These assumptions are fulfilled if the code has been preprocessed by a frontend like the ones discussed in subsections 3.3 and 3.4.

In our approach, we choose to determine the set of states in the EFSM model corresponding to a set of *essential* program counter values (a set of *essential* nodes in the CFG). A transition between two EFSM states corresponds to a path between CFG nodes which contains at least one *relevant* node. (We will detail the concepts of *relevant* and *essential* CFG nodes in Section 4.2).

This is different from the classical approach of defining the states as corresponding to predicates over the state variables, as done in the related approaches in the context of specification mining by static analysis for classes [13], [2]. We have chosen this approach because in real applications all the state variables can be complex data structures and it may be a complex task to determine predicate abstractions in this case.

4.2 Building the EFSM

Relevant Nodes. An important preliminary step consists in identifying the *relevant* nodes of the CFG.

In principle, an aspect is considered to be relevant for our model if it influences the external observable behavior which consists of the messages received or sent by the system.

A variable is marked as *relevant* if one of the following occurs:

- it is on a downstream dataflow from a parameter of a received message
- it is on an upstream dataflow ending in a parameter of a sent message

A CFG node is marked as *relevant* if one of the following occurs:

- it corresponds to a message receive or message send instruction
- it handles a relevant variable

A CFG path is *relevant* if it contains at least one relevant node. Determining the relevant paths is actually a form of program slicing.

Essential Nodes, EFSM States and Transitions. It is not necessary that all relevant CFG nodes (which may be far too many) become states in the EFSM model. We call *essential* nodes only the CFG nodes which correspond to nodes of the EFSM.

We propose the following algorithm to identify the essential nodes and the transitions between them:

- The start node is an essential node, and it corresponds to the initial state of the EFSM.
- Any CFG node containing a ReceiveMesage statement is an essential node. It introduces a new EFSM state. The relevant outgoing paths will correspond to outgoing transitions enabled by the received message. Each of these transitions will end in the next state which will be identified as essential on the respective outgoing path. The relevant path conditions are collected as guard predicates for the corresponding transition, while assignments involving relevant variables are collected as list of actions for the corresponding transition.
- A conditional branching node in the CFG is an essential node only if it uses a relevant variable which has been defined in a node preceding it on an incoming path (this includes also the case of loops). It introduces a new EFSM state which has an incoming transition corresponding to the incoming path with the definition node and outgoing transitions corresponding to the outgoing conditional paths.

After determining the essential nodes and identifying the paths between them which correspond to transitions, for each transition we determine its received messages, guard predicates, actions, sent messages. The guard predicate of a transition is composed of all relevant conditions that are on the corresponding path between the two nodes. The action list of a transition contains all assignment or set operations executed on relevant variables on the corresponding path between the two nodes.

An EFSM is deterministic if from any state s, when any message rm is received, there is at most one transition possible. The EFSM built according to the method presented above is deterministic, since transitions outgoing from a state, in the case that they are labeled with the same received message, they have mutually exclusive guard predicates, since they resulted from different paths of the CFG .

4.3 Example

We consider the Online Shop example. By applying technology speciffic preprocessings, its abstract control flow graph has been obtained. For presentation purpose, we use here pseudocode to describe the abstract control flow.

```
1:   orders:={}
2:   payments:={}
3:   while(true)
4:     switch ReceiveMesssage():
5:      case:(orderType, name)
6:            add name to orders
7:      case:(payType, name)
8:            if (name in orders)
9:               add name to payments
10:     case:(deliveryType, name)
11:           if (name in payments)
12                remove name from payments
13:               remove name from orders
14:               SendMessage
                      deliveryResp, goods
15:          else  SendMessage
                       deliveryResp, error
16:  endwhile
```

We determine the nodes (pseudocode statements) 1 and 4 as being the essential nodes, according to the method outlined before. The five possible execution paths below this node correspond to five self-loop transitions. The resulting EFSM is the one which has been depicted in Figure 2.

5 Related Work

As mentioned in the introductive section, the need to infer models occurs at two different scenarios: at service consumption time, and at service deployment time.

At service consumption time, services are black-boxes that come without (trusted) models and their code is not available. A model can be inferred from I/O sequences. There is a large field of research of learning behavioral models by combining black-box testing and automata learning [9], and it begins to be used for inferring models of web applications [4], [7], [11].

At service deployment time, the implementation code is available and model extraction tools should take advantage of having full access to the code of the implementation. Thus, in this case another category of white-box model inference is needed.

The core of our model extraction approach relates with the work on static analysis in the context of specification mining for classes, such as [13], [2], [6]. Automata-based abstractions are used for behavioral modeling, but, as we mentioned in subsection 4.1, they use predicate abstraction in order to determine the states.

Extracting models of web applications through code analysis has been done only on particular technologies or cases such as in [1], [15], [10], focusing on the detection of concrete problems, not on the extraction of a transferable model to be passed for analysis to existing tools.

Extracting models is the main goal of black-box approaches such as [9], [7]. The focus of these works is mainly on developing learning algorithms, in the context of abstract input and output traces. Most relevant from our perspective are the works of [4], [11] which identified the need of automatizing the learning-setup in order to enable learners to interact directly with real applications. These works propose solutions and tools for abstractizing the input and output alphabet from WSDL web services, based on principles which are similar with the ones presented in Section 3.3.

6 Conclusions

The goal of our work is to build a tool for the automatic extraction of behavioral models from service implementations. In order to cope with the diversity of technologies and APIs which can be used by service implementations, we propose an approach for model extraction in two steps: a technology-dependent preprocessing step, followed by the stable core step that implements a general method of transforming the abstracted control flow graph into an EFSM.

The kind of EFSM inferred by our approach is suitable to be automatically translated into an entity description in a formal security specification language for distributed systems such as Aslan++, the language used by the SPaCIoS tool. The security analyst will have to add manually only the security-related properties of the communication channels, which cannot be known from the implementation code, and to specify the desired properties to be checked.

Having tools which extract behavioral models from actual service implementations is an important step towards enabling formal security validation techniques to be applied on real systems at their implementation and deployment time.

Acknowledgements. This work has been supported by the FP7-ICT-2009-5 project no. 257876 SPaCIoS ("Secure Provision and Consumption in the Internet of Services".)

References

1. Albert, E., Østvold, B.M., Rojas, J.M.: Automated extraction of abstract behavioural models from jms applications. In: Stoelinga, M., Pinger, R. (eds.) FMICS 2012. LNCS, vol. 7437, pp. 16–31. Springer, Heidelberg (2012)
2. Alur, R., Černý, P., Madhusudan, P., Nam, W.: Synthesis of interface specifications for Java classes. In: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2005), pp. 98–109. ACM, New York (2005)

3. Armando, A., Carbone, R., Compagna, L., Li, K., Pellegrino, G.: Model-checking driven security testing of web-based applications. In: 2010 Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW), pp. 361–370 (2010)
4. Bertolino, A., Inverardi, P., Pelliccione, P., Tivoli, M.: Automatic synthesis of behavior protocols for composable web-services. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2009), pp. 141–150. ACM, New York (2009)
5. Buchler, M., Oudinet, J., Pretschner, A.: Semi-automatic security testing of web applications from a secure model. In: 2012 IEEE Sixth International Conference on Software Security and Reliability (SERE), pp. 253–262 (2012)
6. Corbett, J.C., Dwyer, M.B., Hatcliff, J., Laubach, S., Pasareanu, C.S., Robby, Zheng, H.: Bandera: extracting finite-state models from java source code. In: Proceedings of the 2000 International Conference on Software Engineering, pp. 439–448 (2000)
7. Hossen, K., Groz, R., Richier, J.L.: Security vulnerabilities detection using model inference for applications and security protocols. In: IEEE 4th International Conference on Software Testing, Verification and Validation Workshops, pp. 534–536 (2011)
8. IBM. Watson, T.J.: Libraries for Analysis (WALA). Technical report, IBM T.J.Watson Research Centre (2010)
9. Lorenzoli, D., Mariani, L., Pezze, M.: Automatic generation of software behavioral models. In: ACM/IEEE 30th International Conference on Software Engineering (ICSE 2008), pp. 501–510 (2008)
10. Mariani, L., Pezzè, M., Riganelli, O., Santoro, M.: SEIM: static extraction of interaction models. In: Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems (PESOS 2010), pp. 22–28. ACM, New York (2010)
11. Merten, M., Howar, F., Steffen, B., Pellicione, P., Tivoli, M.: Automated inference of models for black box systems based on interface descriptions. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part I. LNCS, vol. 7609, pp. 79–96. Springer, Heidelberg (2012)
12. von Oheimb, D., Mödersheim, S.: ASLan++ — a formal security specification language for distributed systems. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) Formal Methods for Components and Objects. LNCS, vol. 6957, pp. 1–22. Springer, Heidelberg (2011)
13. Shoham, S., Yahav, E., Fink, S.J., Pistoia, M.: Static specification mining using automata-based abstractions. IEEE Transactions on Software Engineering 34(5), 651–666 (2008)
14. Sora, I., Popovici, D.-T.: Extracting behavioral models from service implementations. In: Proceedings of 8th International Conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE 2013), pp. 226–231. SciTePress (2013)
15. Tripp, O., Pistoia, M., Fink, S.J., Sridharan, M., Weisman, O.: TAJ: effective taint analysis of web applications. In: Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2009), pp. 87–97. ACM, New York (2009)
16. Viganò, L.: Towards the secure provision and consumption in the internet of services. In: Fischer-Hübner, S., Katsikas, S., Quirchmayr, G. (eds.) TrustBus 2012. LNCS, vol. 7449, pp. 214–215. Springer, Heidelberg (2012)

Impact-Driven Regression Test Selection for Mainframe Business Systems

Abhishek Dharmapurikar, Benjamin J.R. Wierwille,
Jayashree Ramanthan, and Rajiv Ramnath

Ohio State University, Computer Science and Engineering, Columbus, Ohio, U.S.A.
{dharmapurikar.1,wierwille.3,ramanathan.2,ramnath.6}@osu.edu

Abstract. Software testing is particularly expensive in the case of legacy systems involving mainframes. At the same time these systems are critical to many large enterprises and they are perpetually in costly maintenance. For example, even small changes to the system usually lead to an end-to-end regression test. Also, due to the age of legacy systems there is a lack of essential knowledge (e.g. component inter-dependence) and this results in comprehensive system tests that have to be conducted in production environments. This is called the "retest-all" approach and is done to ensure confidence in the functioning of the system. But this approach is also impractical primarily due to: a) resource needs, and b) user stories generated within the agile processes that require changes to the system at an ever-faster pace. The research reported here is aimed at reducing the required regression testing and the costs associated with the system and its assets. The improvements are achieved by identifying only those tests needed by assets changes and others that are 'impacted'. The impact analysis leverages modern static code analysis tools such as Rational Asset Analyzer and dedicated test environments for mainframes. We show that by using impact analysis on a real-world mainframe application the test savings can be about 34%.

Keywords: Regression Test Selection, Legacy Testing, Software Testing, Mainframes Testing, Software Modernization.

1 Introduction

The legacy systems and mainframes are still being used by many enterprises, but are also constantly changing to meet the evolving needs of modern enterprisees. Typically legacy software goes through evolution activities, which can be divided, into three categories maintenance, modernization, and replacement [3]. Almost immediately after being built the system goes through maintenance activities to keep up with the changing business needs. A modernization effort is then required that represents a greater effort, both in time and functionality, than the maintenance activity. Finally, when the old system can no longer be evolved, it must be replaced. Thus all the mainframe systems that have been modernizing to keep up with needs would immediately benefit from reduced testing costs.

Software testing is the most critical and expensive phase of any software development life cycle. According to Rothermel et al., [5], a product of about 20,000 lines of

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 203–217, 2013.

code requires seven weeks to run all its test cases and costs several hundred thousands of dollars to execute them. Software maintenance activities, on an average, account for as much as two-thirds of the overall software life cycle costs [1]. Among activities performed as part of maintenance, regression testing takes large amounts of time as well as effort, and often accounts for almost half of the software maintenance costs [2]. Regression testing by definition (also referred to as program re-validation) is carried out to ensure that no new errors (called regression errors) have been introduced into previously validated code (i.e., the unmodified parts of the program) [2]. With mainframe systems containing several thousands of programs, usually an end-to-end regression test is carried out using test cases from system tests. This black box testing technique is the only practical way of assuring compliance and owing to the lack of knowledge of dependence among components; it is not possible for the system testers to test only the affected components of the system resulting from a change.

There have been many studies to reduce the cost associated with regression testing. Three techniques, test case reduction, test case prioritization and regression test selection are most prevalent. Test case reduction techniques are aimed to compute a small representative set of test cases by removing the redundant and obsolete test cases from test suites [6], [7], [8], [9], [10], [11]. These techniques are useful when there are constraints on the resources available for running an end-to-end regression. Test case prioritization techniques aim at ranking the test cases execution order so as to defect faults early in the system [5]. It provides a way to find more bugs under a given time constraint, and because faults are detected earlier, developers have more time to fix these bugs and adjust the project schedule. Khan et al., in [12] have given comparison of both the techniques and the effect on software testing. Test case prioritization techniques only prioritize the test cases but do not give a subset of cases which would reveal all the faults in the changed system. Test case reduction techniques do give a reduced number of test cases but the coverage of the reduced test cases spans across the entire system including the parts which were not changed. Also these techniques have been proven to reduce the fault detection capacity of the suites [2]. Regression test selection (RTS) techniques select a subset of valid test cases from an initial test suite (T) to test that the affected but unmodified parts of a program continue to work correctly. Use of an effective RTS technique can help reduce the testing costs in environments in which a program undergoes frequent modifications.

Our technique, Impact-Driven Regression Test Selection (ID-RTS) builds on this idea and aims at reducing test costs for mainframe systems and is proposed as a replacement for the retest-all regression tests. The core contribution is a method to ensure that the tests are i.e. the tests selected should reveal all the modifications done to the system, save costs and increase system availability.

The rest of the paper discusses RTS techniques and ID-RTS. Section 2 discusses the different regression test selection techniques. Section 3 of this paper analyzes the structures of the assets and the dependencies among them to determine safe testing needs. Section 4 describes test case selection through impact analysis. Section 5 analyzes the efficiency of the RTS technique using standardized metrics. Section 6 describes an experiment carried out to gauge the savings from using this technique. Section 7 analyzes the results from the experiment and extrapolates savings for a year for an actual enterprise using real data for changes in that period.

2 Regression Test Selection Techniques

Rothermel and Harrold [13] have formally defined the regression test selection problem as follows: *Let P be an application program and P′ be a modified version of P. Let T be the test suite developed initially for testing P. An RTS technique aims to select a subset of test cases T′ ⊆ T to be executed on P′, such that every error detected when P′ is executed with T is also detected when P′ is executed with T′.*

There have been many techniques presented for regression test selection. Code based techniques (also called program based techniques) look at the code of programs and select relevant regression test cases using control flow, data or control dependence analysis, or by textual analysis of the original and the modified programs.

Dataflow analysis-based RTS techniques explicitly detect definition-use pairs for variables that are affected by program modifications, and select test cases that exercise the paths from the definition of modified variables to their uses [17], [18]. However, these techniques are not safe due to their inability to detect the effect of program modifications that do not cause changes to the dataflow information. Also, they do not consider control dependencies among program elements for selecting regression test cases. As a result, these techniques are unsafe [16], [4].

Control flow techniques [21], [22] model control flow of input programs into control flow graphs and analyze a change on them for selecting regression test cases. These techniques have been proven safe and the graph walk approach suggested in [22] is the most precise work for procedural languages [16] and most widely used control flow technique [27]. However they do not include non-code based components of the system such as DB and files.

Dependence based RTS techniques look at the data and control dependencies among or within the programs to filter out the modified test cases. Program dependence graph based technique suggested in [19] could work on a single program, which was later improved to system wide scope in [20] by using System dependence graph, but both the techniques were proved to be unsafe as they omit tests that reveal deletions of components or code [4].

Differencing technique [19] is an RTS technique based on textual differencing of the canonical form of the original and the modified programs. Though this technique is safe, it requires conversion of programs into a canonical form and is highly language dependent. Also the complexity of this approach is too high to be feasible for a mainframe system with several thousand programs [16].

Slicing based techniques [23] select those test cases which can produce different outputs when executed with the modified program version P. Agarwal et al.[23] have defined 4 different slicing techniques. The basic slicing technique forms an execution slice which is the set of program statements in program P that are executed for a test case t. Other three techniques build on the basic technique picking statements that influence an output, or predicate statements that affect the output. The overall technique would select a test case t only if the slice of t computed using any one of the four approaches contains a statement modified in P. These techniques are precise, however they have been shown to omit modification-revealing tests, hence are not safe [16].

The most relevant research to ID-RTS is the firewall based approach in [24]. A firewall is defined as a set of all modified modules in a program along with those

modules which interact with the modified modules. This technique uses a call graph to establish control flow dependencies among the modules. Within the firewall, unit tests are selected for the modified modules and integration tests for the interacting modules. This technique is safe as long as the test suite is reliable [4].

Research has also been done on specification based regression test selection techniques which look at the specification of a program by modeling the behavior [14] and/or requirements of a system [15]. These techniques do not employ the dependency extracted from static code analysis of programs and hence are not precise or safe [16].

3 Mainframe Asset Structures and Dependencies

Impact-Driven Regression Test Selection (ID-RTS) defined here is a control flow and data dependence based, intra-procedural regression test selection technique designed for mainframes. It filters out test cases based on the following steps

1. Represent in a comprehensive way all the inter-asset dependencies in a dependence graph by static code analysis.
2. Analyze the types of changes to the system. This involves accounting for insertion, modification or deletion of an asset.
3. Filter out the affected interfaces and associated test cases for a system through impact analysis for a particular change.

In order to make the dependence based RTS safe, all dependencies within the mainframe system must be represented.

The main language that runs on mainframes is COBOL (COmmon Business-Oriented Language), originally consisted of source programs, copybooks, JCLs, PROC files and record oriented files. Mainframe systems have evolved overtime to support many modern features such as relational databases, multiple file systems and layouts, transaction and information management systems etc. The source for all components would form *assets* of the system, which consist of files, source programs, database tables and batch jobs. This section would highlight the dependencies that would exist among the various assets.

N. Wilde in [26], has listed out all the possible dependencies that can exist among and within programs. The concepts mentioned can be extended to represent the dependencies amongst the assets in mainframes.

The topics covered next describe the possible data and control dependencies that could exist among the assets to form the dependence graph and that have to be considered to make the testing safe. In general, *assets are represented by the nodes in the graph and the dependencies by the edges between them.* This graph is used to analyze the impact of a change on any assets for safe testing.

3.1 Source - Copybook Dependencies

As mentioned before, copybooks contain data structure variables that would be used in the source programs. COPY statements are used inside the programs to include and

use these data structures inside COBOL programs. The compiler expands the copybooks inline inside the programs, so that the references are resolved.

To establish dependencies among the copybooks and programs the definition-usage model proposed by the dataflow methods is used. If a variable is defined inside the copybook and is used inside a program a dependency exists and an edge between the program and the copybook exists in the dependence graph.

As a good programming practice, due to the inline expansion of the copybooks inside the code, it is advisable to design copybooks such that they contain minimum number of structures that several programs would use. For e.g. Copybook A contains definition of variables, and *varb* and program *prga* includes this copybook but uses only *vara* and similarly program *prgb* uses only *varb*. The copybook would expand inline and the programs will end up having unused variables inside. Ideally there should be two copybooks defined each for *vara* and *varb* and only the required copybook should be included by the programs. With such a practice, only those programs impacted by a change in the copybook file can be extracted. However these practices cannot be imposed upon programs that have already been written and are running in production.

The dependencies are hence established at the data structure level, i.e. instead of the entire copybook, the data structures inside it would now form nodes in the graph. An edge is drawn from the source program to a node if that data structure is used by the program (E.g. PGM1 – VAR2 dependency in Figure 1).

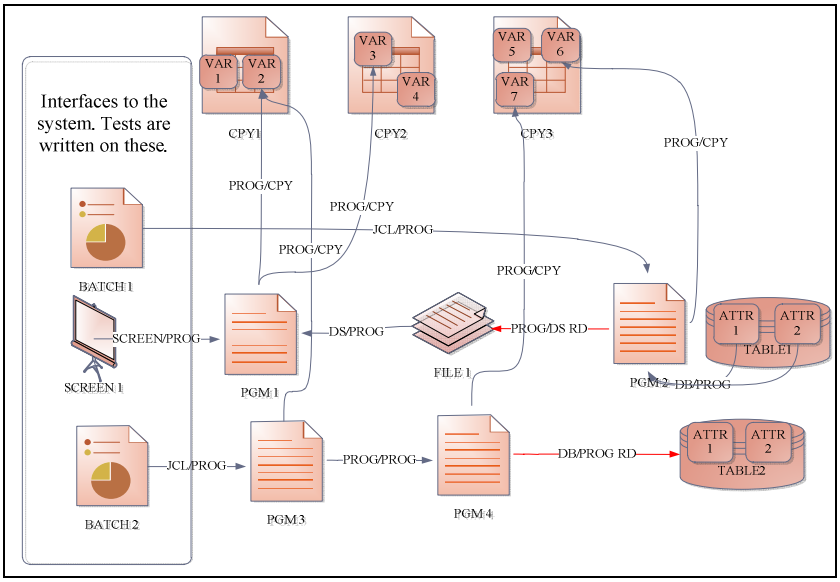


Fig. 1. The dependencies amongst various assets in a mainframe system. The batch jobs and online screens form the interfaces to the system. The system tests test these assets.

3.2 Source – Source Dependencies

COBOL programs can call other programs through the CALL statement. The programs can be called by directly using the program name as a literal or using an identifier contained in a constant or variable. If a program A calls another program B it creates a dependency on B which is represented by an edge in the graph (E.g. PGM3 – PGM4 dependency in Figure 1).

If a program is called using its name stored in a variable whose value is dynamically populated during the execution of the calling program, the dependencies amongst the programs cannot be determined through static analysis. For the scope of this research, coding best practices should be established to avoid dynamic calls. We have verified that the system under test does not have any such dynamic calls.

3.3 Source – File (Dataset) Dependencies

COBOL programs use a file as intermediate output between two programs or as terminal output. Files are opened in INPUT, OUTPUT or EXTEND mode which corresponds to file read, write or append respectively. If the program opens the file in read mode then the program is dependent on the file (E.g. PGM2 – FILE1 dependency in Figure 1) and if it opens in write mode, the file depends on the program file (E.g. FILE1-PGM1 dependency in Figure 1). This helps to establish a transitive dependency between programs that write to a file and those that read it (PGM2 is transitively dependent on PGM1). Both control files and data files can be represented using the same model.

JCL 'DD' statements are used to identify files that the program will reference. The function of a DD Statement is to form a logical connection between an actual file and an identifier a COBOL program will use to refer to that file. The file based dependency can also be associated with the JCLs instead of the source without any change in the impact analysis. However for the scope of this paper the former approach is taken.

3.4 Source – Database Dependencies

As with source programs and files, dependencies also exist between the programs and databases. COBOL uses EXEC SQL statement to embed SQL queries into the source programs. Similar to files, tables can be read from or written to. From the way tables are accessed in a program a dependency can be established. Update, insert or delete queries are written to the database making the table dependent on the program. Select queries make the program dependent on the database.

However, a program might not use all the attributes from the table. The dependency has to be classified into two types. One dependency when the program accesses all attributes from the table using a '*' in the SQL statement, in which case the program is dependent on the entire table (E.g. TABLE2-PGM4 dependency in Figure 1). Any change in the structure of the table would affect these programs. Other dependency arises when the program accesses limited attributes from the table. Such dependency relations have to be maintained at the attribute level (E.g. the dependency between

ATTR1 and PGM2 in Figure 1). DDL queries do not create any dependencies as they are not executed along with the transactions in the system. Hence, execution of DDLs is treated as changes to the database and impact analysis is carried out on the affected tables and attributes as discussed later in Section 4.

Stored procedures can be treated as programs that have SQL statements to manipulate the DB, and hence this creates a source to DB dependency. Source to Source dependencies would exist between the calling program and the stored procedures. DDL queries do not create any dependencies as they are not executed along with the transactions in the system. Hence, execution of DDLs is treated as changes to the database and impact analysis is carried out on the affected tables and attributes as discussed later in Section 3.

3.5 Jcl – Source Dependencies

JCLs are used to run COBOL programs in batch mode. They contain steps that execute PROCs or programs and report the return code of the execution indicating if the job has failed or passed. JCLs are dependent on a program if in any step they are executing that program (E.g. BATCH1 – PGM2 dependency in Figure 1). If a PROC is executed in any step the JCL is then dependent on the PROC, and in turn the PROCs are dependent on the programs they execute.

3.6 Screen – Source Dependencies

Screens are like programs, but can be executed online in a transaction. They can be treated as a program and have same dependencies as a program would have (E.g. SCREEN1 - PGM1 dependency in Figure 1).

3.7 Copybook - Copybook Dependencies

COPY statements can exist within copybooks too. If a copybook A is copied by another copybook B, all variables in B using variables in A are dependent on the used variables. To preserve the variable level granularity in copybooks, the dependency is mapped between the variable's definition and declaration.

There are several other types of assets that can exist in the mainframe system, for e.g. report writer programs, assembler programs etc. These assets can be categorized into types that were discussed and dependency rules of that type can be applied to them. In general, i) if an asset A uses code from or transfers control to an asset B, then A is said to be dependent on B, ii) If an asset A writes data into an asset B, then B is dependent on A. On the other hand, if an asset A reads data from an asset B, then A is said to be dependent on B.

To any mainframe system, batch jobs and online transactions act as interfaces. These batch jobs and transactions give a certain output based on the input provided. The output can be written to the screen of the transaction or to a file or database changing the state of the system. System test cases are run against all these interfaces to poll for the outputs corresponding to the inputs to be tested (see Figure 1).

4 Filtering Interfaces and Tests

By doing a static analysis of all the assets in the system, a system dependence graph G is created. Changes to the system are then analyzed to filter any impacts on the interfaces. This is done by graph traversal starting from the seed/s of change/s to the interfaces on the inverse (G^{-1}) of the graph G. The interfaces touched by the graph traversal will form the set of affected interfaces. The system tests associated with the interfaces are filtered as part of test selection (see Figure 2). These tests are run on the updated system to check for compliance in a test environment. Prior to the filtering, the test cases must be updated to reflect the change. Once all the tests pass, the graph G is updated according to the changes made to the assets.

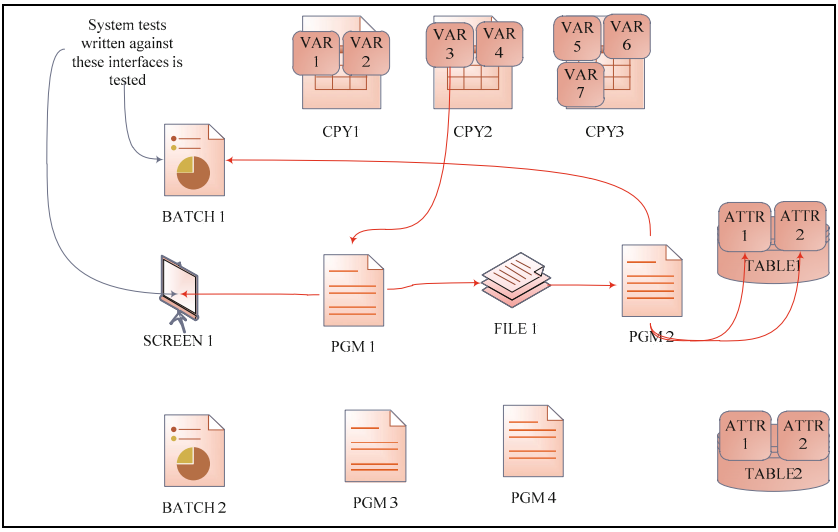


Fig. 2. A depiction of impact of change. The VAR3 data structure variables changed, whose impact is carried over the interfaces through graph traversal on the inverse graph G in Figure 1.

Addition of code to existing programs can be dealt as modification of that asset, and the graph traversals can be started from the changed asset as a seed. However, new assets created (or new attributes for tables or new variables for copybooks), does not affect the system unless they are used. E.g. If a new program A is added, it will not affect the system unless it is called by some other programs, JCLs or PROCs. The assets where the new additions are used form the seeds of change for graph traversal and the affected interfaces are then filtered.

Like the firewall technique in [24], ID-RTS filters out test cases based on the modules that interact with the change. However, instead of filtering out integration tests on the first level modules that interact with the change, ID-RTS filters the system test cases at the entry points to the system that directly or transitively interact with the change. With mainframes, at least with the system in test, the unit or integration test cases are not properly defined as these test techniques were not widespread at the beginning of

mainframe development. Moreover, programs cannot be tested standalone without a batch job submitting them. Today, mainframes are being tested against the designed system tests which mandate the behavior of the system. If tests and techniques enable fine grained tests (like unit or integration), ID-RTS can be extended to analyze impact on the assets that have tests available for and select test cases accordingly.

5 Analysis

Harrold et al., in [5] have given analysis metrics to gauge the effectiveness of any RTS techniques. They have defined a test t to be modification-revealing if the output of the case differs in the original (P) and the modified system (P′). The metrics identified were *Inclusiveness, Precision, Efficiency* and *Generality*.

Inclusiveness measures the extent to which an RTS technique selects modification-revealing tests from the initial regression test suite T. Let us consider an initial test suite T containing n modification-revealing test cases. If an RTS technique M selects m of these test cases, the inclusiveness of the RTS technique M with respect to P, P′ and T is expressed as (m/n) ∗ 100. A safe RTS technique selects all those test cases from the initial test suite that are modification-revealing. Therefore, an RTS technique is said to be safe, iff it is 100% inclusive.

Harrold et al., in [5] have also defined a test to be modification-traversing. A test t is modification-traversing if it executes all the modified and deleted code, irrespective of the output given. A set of modification-traversing tests is the superset of modification-revealing tests. The ID-RTS approach filters out test cases that traverse more than the modified assets, giving 100% inclusiveness and safety.

The primary aim of ID-RTS is to be safe, as the approach plans to replace the existing retest-all techniques. The rationale behind any retest-all technique is to gain confidence on the compliance of the system behavior. ID-RTS works on this requirement and identifies the assets that are impacted by the change and includes all the tests associated with the impacted assets. Like the firewall technique [24], ID-RTS needs the system tests to be reliable. However, for mainframes, they are reliable because the same tests are used to guarantee compliance with the existing retest-all approach.

Precision measures the extent to which an RTS algorithm ignores test cases that are non-modification-revealing. Suppose T contains n tests that are non-modification-revealing for P and P′ and suppose M omits m of these tests. The precision of M relative to P, P′ and T is given by the expression (m/n) ∗ 100 or 100% if n = 0.

As explained earlier, in any system test the test suite contains only the tests that run at the entry points (interfaces) of the system. In evaluating the precision of ID-RTS, there would be no unit tests in T. Even with that said, due to the coarse inter-procedural level filtering that ID-RTS employs, it is not precise. Also, Rothermel in [25], has shown that the savings acquired from fine grained intra-procedural techniques may not justify the costs associated. With the system under test, which is a typical mainframe system, the large number of programs would have even higher costs for using intra-procedural techniques.

Also, precision varies with the modularity of the system. If the system is designed such that there exists more number of small programs, copybooks and transactions, inter-procedural test selection can also be fine grained. As the impact is flooded from the seed, only programs that would indeed by affected by the change are filtered, eventually selecting only the modification-revealing interfaces. Coding best practices can be established to have modularity in the system, but the costs of changing the existing code base vs. the cost of testing extra test cases is highly debatable.

Efficiency measures the time and space requirements of the RTS algorithm. Let P denote programs and PROCs, V denote sum of all data variables in all copybooks (a record structure counts as a single variable), A denote all database table attributes, F denote files, J denote JCLs and screens, the complexity can be given as $O(P^2 + VP + AP + JP + FP)$. The space complexity is also of the same order.

Generality is the ability of the technique to work in various situations. ID-RTS was designed to work only in the mainframe environment and cannot be generically applied to other systems as is. However, it provides a basis for designing a framework for other business oriented languages.

6 Experiment Setup

To calculate the savings from using ID-RTS, we conducted a small experiment to analyze a real world mainframe application for dependencies and impacts of changes. IBM's Rational Asset Analyzer (RAA), a static analysis tool available for mainframes was used. RAA statically analyzes all the assets imported into the RAA server and establishes dependencies among them as described earlier. From the dependencies established, it also analyzes impact of a change in source programs, data elements in copybooks, other files and DB2 DB. As a contribution, this is one of the few studies to have been actually designed and tested in an enterprise environment [27].

As only the interfaces are tested in systems tests for an application, for the experiment, the impacted interfaces were filtered for each change, instead of filtering the actual test cases. This also assumes that tests are evenly distributed across the interfaces. Also, as CICS screens and transactions were not imported into RAA for the application under test, only batch jobs were considered. But this does not affect the generality of the research, for RAA analyzes impact of a change on all assets of the system including screens and transactions.

The nature of the applications under test is similar to the depiction in Figure 1. App A has around 6,707 assets in all containing 2,287 source programs, 1,554 JCL batch jobs and 1,823 copybooks. The rest form the control and data files. This excludes the CICS transactions and screens that the application might use. As this application does not use any database, we tested another application, App B, for impact of database changes. App B has 48,210 assets in all with 109 DB2 database tables and 5,393 JCL batch jobs.

To calculate the savings from implementing ID-RTS we have taken a two-step approach,

1. We identify the impact of last 2 changes in each asset type in set A (namely JCLs, programs, copybooks, files and DB tables) on the interfaces of the system. From the data collected, the mean impact by asset type (MIT) is calculated as % of the 1,554 (5,393 for App B) interfaces affected. As the impact of an asset change depends on the number of other assets dependent (directly or transitively) on it, we expect the number of impacted interfaces per asset type to be in the decreasing order for copybooks, programs, files, DB and JCLs for this particular system.
2. To calculate the actual savings, we first calculate the frequency of occurrence of changes by asset type (FT) as % in an agile iteration and then extrapolate impacts using the weighted mean of MIT with respect to FT for each asset type in A. Weighted mean impacted interfaces (WMI) is calculated as

$$WMI = \sum_{\in A} MIT \times FT$$

This would give us the % interfaces we need to test for each change in an iteration. **Savings in % would be 100 – WMI**.

7 Results

We ran the impact analysis of last 2 production changes of each asset type, in order to gauge the efficiency of ID-RTS in real world scenarios. We then filtered out the affected batch jobs from the impact analysis report and calculated the Mean Impact by type as shown in Table 1. As we found that the other RTS techniques as described in section 2 were infeasible or not safe for mainframes, we compare ID-RTS with the existing retest-all technique.

The order of the impacts amongst the asset types was found to be as expected, with the exception of copybooks and source programs. Contrary to our expectations, the impact of the copybook changes was found to be less than that of source programs, however with a close difference. This could be attributed to the nature of assets that were changed; the source programs changed were called by more assets than the programs that used the changed copybooks. This outcome is highly peculiar and moreover, we expected the difference in impacts of changes in copybooks and source programs to be minimal, which was exemplified.

Between files and databases, the outcome of the order was in compliance with the expectations. The order could vary from application to application. For applications

Table 1. Mean Impact by Type for last 2 changes

Asset Type	Mean Impact by Type (%)
JCL	0.06
Database	0.93
Files	43.59
Copybooks	65.54
Source Programs	65.95

under test, the DB2 database tables were introduced in App B much later than the usage of files in App A, accounting to files being used significantly more than the databases. The impact of change is proportional.

The time required for impact analysis varied by asset type, with copybooks and source programs taking the longest, approximately 2 hours on an average, files took 1 hour and 10 minutes and database tables took 10 minutes. JCL impact analysis is not supported in RAA as there are no dependencies on them. For the purpose of the experiment we assumed the impact of a JCL batch job change on interfaces as 1 to account for the same JCL changed. RAA's impact analysis gives a thorough report of all impacted assets, not just the interfaces. This adds to the total time required for impact analysis. Tools that would solely report the impacted interfaces would have significant time savings.

Table 2. The changes done to the App A by type for the period of March 12 - February 2013. Column ID-RTS gives the extrapolated impacted interfaces using data from Table 1 and Restest-all gives the total interfaces in App A to be tested with that technique.

Period	Source Programs	Files	Copy-books	JCLs/Screens	DB	Total	ID-RTS	Retest-all
Mar-12	877	5	32	0	0	914	1022.7	1554
Apr-12	297	0	20	0	0	317	1024.5	1554
May-12	178	0	0	0	0	178	1024.9	1554
Jun-12	324	1	12	0	0	337	1023.6	1554
Jul-12	390	2	6	0	0	398	1023	1554
Aug-12	152	0	0	0	0	152	1024.9	1554
Sep-12	117	0	0	0	0	117	1024.9	1554
Oct-12	656	4	23	0	0	683	1022.6	1554
Nov-12	445	0	21	0	0	466	1024.6	1554
Dec-12	127	0	3	0	0	130	1024.7	1554
Feb-13	141	3	13	0	0	157	1017.7	1554
Total	3704	15	130	0	0	3849	1023.3	1554

To calculate actual savings from ID-RTS, we first recorded changes for the period of March 2012 to February 2013 by asset type. The findings are tabulated in table 2. We then extrapolated impacts on App A by taking weighted mean of MIT with respect to the frequency of change. The results are shown in ID-RTS column of table 2. As, in all iterations programs were changed the most, the impacted interfaces were around 65% of the total 1,554 interfaces. As per ID-RTS, these are the interfaces to be tested for each change while the retest-all technique tests all of them. For the entire period the average impacted interfaces per change were 1,023.3 (65.85% of the 1,554 interfaces). Thus the ID-RTS technique can save approximately **34%** of testing efforts.

This result can vary by the nature of dependencies within the application and the types of changes that are done. If for App A, there are more changes to DB, files and JCLs than copybooks and programs the savings would be proportionally more. Also, the impact might change in assets of the same type, depending on the nature of dependencies. E.g. Program A has more dependencies than program B, the impact of change of A would be proportionally more. This experiment was conducted to find out if the impact of actual production changes span to a subset of the system and estimate savings for ID-RTS. Hence we also limited the test to last 2 changes to each asset type. Also, a learning phase can be planned, where the impacts of production changes are monitored to gauge the inter-dependency among the assets of the system. If all the changes impact significant percentage of interfaces, the retest-all technique can be employed for that system, saving the time required for impact analysis.

8 Conclusions and Future Work

The retest-all system test technique which tests all components of the system for regression test can be replaced by our proposed Impact-Driven RTS. Modern analysis tools such as IBM's RAA can be used to draw dependencies among assets of the system and analyze impact of a change. The impact of a change spans to a subset of the system, providing significant savings in the test cycle times which reduces associated costs and increases system availability.

As future work, the various studies on the object oriented RTS techniques can be integrated to design the framework for object oriented COBOL. Also, the current framework can be extended to include other business oriented systems such as SAP-ABAP. Similar to mainframes these systems use batch jobs, online transactions, DBs and files within a single system boundary.

References

1. Pressman, R.: Software Engineering: A Practitioner's Approach. McGraw-Hill, New York (2002)
2. Leung, H., White, L.: Insights into regression testing. In: Proceedings of the Conference on Software Maintenance, pp. 60–69 (1989)
3. Weiderman, N.H., Bergey, J.K., Smith, D.B., Tilley, S.R.: Approaches to Legacy System Evolution. In (CMU/SEI-97-TR-014) Pittsburgh, Pa. Software Engineering Institute, Carnegie Mellon University (1997)
4. Rothermel, G., Harrold, M.: Analyzing regression test selection techniques. IEEE Transactions on Software Engineering, 529–551 (August 1996)
5. Rothermel, G., Untch, R.H., Harrold, M.J.: Prioritizing test cases for regression testing. IEEE Trans. on Software Eng. 27(10), 929–948 (2001)
6. Harrold, M.J., Gupta, R., Soffa, M.L.: A methodology for controlling the size of test suite. ACM Trans. on Software Eng. and Methodology (TOSEM), 270–285 (1993)
7. Hennessy, M., Power, J.F.: An analysis of rule coverage as a criterion in generating minimal test suites for grammar based software. In: Proceedings of the 20th IEEE/ACM In-

ternational Conference on Automated Software Engineering (ASE 2005), Long Beach, CA, USA, pp. 104–113 (November 2005)
8. Kandel, P.S., Last, M.: Test cases generation and reduction by automated input-output analysis. In: Proceedings of 2003 IEEE International Conference on Systems, Man and Cybernetics (ICSMC 2023), Washington, D.C., vol. 1, pp. 768–773 (October 2003)
9. Vaysburg, L.H.T., Korel, B.: Dependence analysis in reduction of requirement based test suites. In: Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2002), Roma, Italy, pp. 107–111 (2002)
10. Jones, J.A., Harrold, M.J.: Test-suite reduction and prioritization for modified condition/decision coverage. IEEE Trans. on Software Engineering (TSE 2003) 29(3), 195–209 (2003)
11. Jeffrey, D., Gupta, N.: Test suite reduction with selective redundancy. In: Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM 2005), Budapest, Hungary, pp. 549–558 (September 2005)
12. Khan, S.R., Rahman, I., Malik, S.R.: The Impact of Test Case Reduction and Prioritization on Software Testing Effectiveness. In: International Conference on Emerging Technologies, pp. 416–421 (October 2009)
13. Rothermel, G., Harrold, M.: Selecting tests and identifying test coverage requirements for modified software. In: Proceedings of the International Symposium on Software Testing and Analysis, pp. 169–184 (August 1994)
14. Chen, Y., Probert, R., Sims, D.: Specification based regression test selection with risk analysis. In: CASCON 2002 Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research, p. 1 (2002)
15. Chittimalli, P., Harrold, M.: Regression test selection on system requirements. In: ISEC 2008 Proceedings of the 1st Conference on India Software Engineering Conference, pp. 87–96 (February 2008)
16. Biswas, S., Mall, R., Satpathy, M., Sukumaran, S.: Regression Test Selection Techniques: A Survey. Informatica 35(3), 289–321 (2011)
17. Harrold, M., Soffa, M.: An incremental approach o unit testing during maintenance. In: Proceedings of the International Conference on Software Maintenance, pp. 362–367 (October 1988)
18. Taha, S.T., Liu, S.: An approach tosoftware fault localization and revalidation based on incremental data flow analysis. In: Proceedings of the 13th Annual International Computer Software and Applications Conference, pp. 527–534 (September 1989)
19. Vokolos, F., Frankl, P.: Pythia: A regression test selection tool based on textual differencing. In: Proceedings of the 3rd International Conference on Reliability, Quality & Safety of Software-Intensive Systems (ENCRESS 1997), pp. 3–21 (May 1997)
20. Ferrante, J., Ottenstein, K., Warren, J.: The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems 9(3), 319–349 (1987)
21. Laski, J., Szermer, W.: Identification of program modifications and its applications in software maintenance. In: Proceedings of the Conference on Software Maintenance, pp. 282–290 (November 1992)
22. Rothermel, G., Harrold, M.: A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology 6(2), 173–210 (1997)
23. Agrawal, H., Horgan, J., Krauser, E., London, S.: Incremental regression testing. In: IEEE International Conference on Software Maintenance, pp. 348–357 (1993)

24. Leung, H., White, L.: A study of integration testing and software regression at the integration level. In: Proceedings of the Conference on Software Maintenance, pp. 290–300 (November 1990)
25. Rothermel, G.: Efficient, Effective Regression Testing Using Safe Test Selection Techniques. PhD dissertation, Clemson Univ. (May 1996)
26. Wilde, N.: Understanding Program Dependencies. In: SEI-CM (August 1990)
27. Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: a survey. Journal of Software Testing, Verification and Reliability 22(2), 67–120 (2012)

Improving Business Process Model after Reverse Engineering

María Fernández-Ropero, Ricardo Pérez-Castillo, and Mario Piattini

Instituto de Tecnologías y Sistemas de la Información, University of Castilla-La Mancha,
Paseo de la Universidad 4, 13071, Ciudad Real, Spain
{marias.fernandez,ricardo.pdelcastillo,mario.piattini}@uclm.es

Abstract. An appropriate business process management helps companies to quickly adapt to changes while their competitiveness is maintained or even improved. For this reason, business process models are recognized as being important assets for companies and companies are currently demanding mechanisms to ensure business processes with an appropriate quality degree. These business process models can be mined by reverse engineering from existing information systems. Regrettably, these reversed models usually have a lower quality degree and may not exactly reflect the actual business processes. This paper describes detected common problems in business processes retrieved by reverse engineering (e.g., missing, fine-grained or non-relevant elements, ambiguities, etc.). Refactoring is widely-used to fix quality problems in business process models. Despite this, this work suggests addressing the above problems by defining three stages: repairing, refactoring and expert-based improvement. Additionally, some preliminary results from refactoring stage are provided using real-life retrieved business process models.

Keywords: Business Process Model, Business Process Improvement, Business Process Challenges, Refactoring, Understandability, Modifiability, Reverse Engineering.

1 Introduction

Business process management allows organizations to be more efficient, more effective and more readily adaptable to changes than traditional management approaches. Business processes depict sequences of coordinated business activities as well as the involved roles and resources that organizations carry out to achieve their common business goal [1]. They are recognized as one of the most important assets in an organization due to the competitive advantages that they provide for organizations [2]. In order to supply the management of business processes they can be represented by models following standard notations such as BPMN (Business Process Modeling and Notation) [3].

However, organizations may not have their business process models explicitly or aligned with current behavior. In these cases, reverse engineering can be used to mine business process model from existing information system [4]. Nevertheless, the retrieved business process models by reverse engineering entail some problems that can

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 218–228, 2013.

affect to their quality degree since every reverse engineering technique implies a semantic loss [5]. Despite the fact that much academic literature is devoted to identify challenges presented in business process model discovered by mining process (e.g., using event logs [6]) or by hand [7], there are no identified challenges to address in those business process model retrieved from existing information system, for example, from source code. This kind of business process models can be incomplete or can contain non-relevant information, or even may contain ambiguities or uncertainties that decrease their understandability and, therefore, their quality degree. In these cases, it is necessary to improve business process model with the aim to address these quality challenges while making it as similar as possible to the reality that they represent [8].

For this reason, this paper presents a set of challenges detected in business process models obtained from reverse engineering. These challenges are been collected after a literature review and practical experiences with business process models mined from several real-life information systems. With the purpose to have several retrieved business process models to analyze, MARBLE [4], a reverse engineering approach and tool, has been selected to mine them. Moreover, the paper introduces an approach to address the above challenges. The proposal combines reverse engineering with other analysis approaches in order to mitigate the semantic loss that reverse engineering techniques entails. This issue is due to some reverse engineering techniques are focused on source code and there are more knowledge sources from which to extract knowledge. Hence, the approach is divided in three stages: repairing, refactoring and expert-based improvement. Each stage uses additional knowledge (such as recorded event logs, guidelines, heuristics, and expert decision, among other) to improve the business process model. Despite refactoring techniques are the most widely-used solution to improve the quality degree of business process models [9, 10], this work also proposes other two additional stages in order to assist refactoring and enhance business process models. The work also presents some preliminary results achieved by using the proposed approach.

The remainder of the paper is organized as follows: Section 2 introduces the related works. Section 3 summarizes the challenges that retrieved business process models involve. After that, Section 4 introduces the proposed approach in an attempt to address these challenges along three stages. Afterwards, some results obtained by using the proposed approach will be shown in Section 5. Particularly, results obtained after refactoring stage are provided. Finally, conclusions and future works are discussed in Section 6.

2 Related Works

Improvement of business process models have been discussed by several authors in the last years with the aim to fix quality faults. In fact, several techniques such as merging, mining, refactoring or re-use are been provided by Dijkman et al., [10] in order to increase the quality degree of business process models. Particularly, refactoring is the most wide-used by authors in literature. For example, Weber et al., [11] proposes a catalogue of *bad smells* for the identification of refactoring opportunities. Similarly, Dijkman et al., [12] show a metrics-based technique for detecting

refactoring opportunities. La Rosa et al., [13], in turn, are devoted to identify patterns for reducing model complexity through, among other ways, compacting, compositing, and merging. Dumas et al., [14] and Ekanayake et al., [15], for their part, focus on detection of duplicate fragments. Pittke et al., [16] focus on labels and define an mechanism to identify synonym and homonym labels in models repositories. Other works like [17-20] focus on identifying coarse-grained activities by means of business process abstraction to exclude non-relevant information.

All the above approaches identify challenges of business process model but they are focus on business process model obtained by mining process (e.g., using event logs [6]) or by hand [7]. Therefore, none of them attempts to identify and address quality challenges in retrieved business process models by means reverse engineering.

To address this very issue, this paper identifies challenges in retrieved business process models and proposes an approach for addressing these challenges.

3 Challenges in Retrieved Business Process Models

This section presents the challenges to address the most common problems identified in the business process models obtained through reverse engineering. These challenges are been collected after a literature review and practical experiences with business process models mined from several real-life information systems. The selected tool to mine business process model was MARBLE. This tool is an adaptive framework to recover the underlying business process models from legacy information system using source code [4]. MARBLE has been applied to several industrial case studies to recover business processes from a wide variety of legacy information systems. The conduction of these industrial case studies has enabled the tool to be improved and the MARBLE technique to be refined. So far, MARBLE has been used with six legacy systems in all: (i) a system managing a Spanish author organization; (ii) an open source CRM (Customer Relationship Management) system; (iii) an enterprise information system from the water and waste industry; (iv) an e-government system used in a Spanish local e-administration; (v) a high school LMS (Learning Management System); and finally (vi) an oncological evaluation system used in Austrian hospitals [21]. All business process models obtained in each case study (from each of the six systems) were analyzed by experts in order to figure out common errors that frequently occur. Challenges that retrieved business process models entail are collected in the following subsections.

3.1 Completeness

Business process models mined by reverse engineering models may not be fully complete due to the data can be distributed in several sources, not just at the source code itself and therefore they cannot be obtained solely through a static analysis. Business process models may have missed nodes such as business tasks, gateways, events and data objects, as well as missed connections such as sequence flows (between tasks) and association flows (between tasks and data objects). This loss affects the semantic completeness of the model [22]. All these missing elements may not have been instantiated at design time and, for that reason, may not be appear in the business

process model. As a consequence, one of the biggest challenges is to rediscover elements that were not recovered in the reverse engineering phase, as well as the order among different business activities. The order between activities is a very issue since complete sequence flows between activities may not be provided through reverse engineering due to the fact that not all information can be automatically derived from source code. The start points and end may not have been defined in the model because there is not enough information to determine which activities are the beginning or ending of a model or what task is executed before another [23]. An example of this scenario is shown in Fig. 1, where the fragment (a) is related to the real retrieved business process model while the fragment (b) is related to the expected business process model, i.e., with all elements connected.

3.2 Granularity

According to the approach proposed by Zou et al., [24], each callable unit in an information system is considered a candidate business task to be discovered by reverse engineering. However, existing information systems typically contain thousands of callable units of different sizes that are usually considered business tasks that can have different levels of granularity [6, 23] such as (1) large callable units that support the main business functionalities of the system (e.g., methods and functions of the domain and controller layer), (2) small callable units as getter and setter methods in object-oriented programming that only read and write program variables but perform no real business task, (3) a set of small callable units that have similar behavior and perform a business task jointly, or (4) a set of small callable units that can together support another. In that case, the main task may be considered as father task while small tasks may be considered as children tasks. This last scenario is shown graphically in Fig. 2, where Task t1, Task tn, among others tasks, support the Task T.

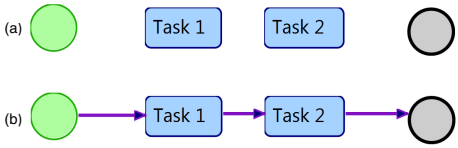


Fig. 1. Incomplete retrieved business process model (a) and its expected version (b)

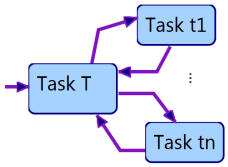


Fig. 2. Retrieved business process model with small tasks that support another task

With the purpose to address this challenge, a solution could be taking into account only coarse-grained callable units as candidate to be business tasks while fine-grained ones are discarded since fine-grained granularity makes models closer to source code perspective. Nevertheless, the dividing line between coarse- and fine-grained callable

units is unknown. Authors like *Polyvyanyy et al.* [19] propose to abstract these business process models to reduce unwanted details and to represent only the relevant information. These authors address the issue of different types of granularity by proposing two techniques: (1) eliminating those small tasks that are considered as irrelevant and (2) grouping certain tasks into one while the information is preserved.

3.3 Relevance

In contrast with completeness which is related to missing elements, relevance is related to business elements that have been retrieved erroneously. The information is considerate as non-relevant when it can be removed without losing information, preserving the behavior. This information (such as activities, events, etc.) may have been created in compilation time but is not used in execution time, i.e., these elements do not carry out any business logic in the organization. Fig. 3 shows a sample fragment of a retrieved business process model with an unnecessary gateway. In that case, the gateway connects one input and one output so that the connection between Task1 and Task2 can be directly in that case due to gateways are designed for use in branches with two or more elements.

The relevance of a business process model is an important aspect since it ensures the model contains enough elements to convey their information [22]. The challenge must be addressed by identifying and removing all non-relevant elements in the business process model while preserving semantics of the relevant parts.

3.4 Uncertainty

The enhancement of the understandability of a business process model is a challenge given that poor understandability of the model can lead to a wrong conclusion. Understandability is usually worse in those models that have been obtained by reverse engineering from existing information systems since identifiers and names of elements may not be enough descriptive. This is because many identifiers are inherited from the elements in source code. For example, task labels usually consist of the concatenation of various capitalized words according to naming conventions present in most programming approaches (see Fig. 4). This kind of names is uncertain but can provide a clue to find more representative task names.

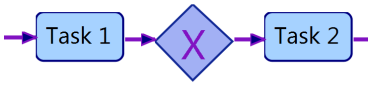


Fig. 3. Retrieved business process model with unnecessary gateway

Fig. 4. Retrieved business task labeled using the CamelCase [25] naming convention

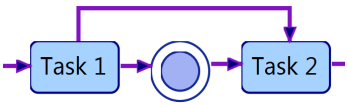


Fig. 5. Ambiguous retrieved business process model

This issue is focused on the interpretability from a language-usage perspective, i.e. how intuitive is the language used to define the elements of the model. For this reason, the labeling of elements can negatively affect the model interpretation when it does not follow an appropriate convention [22]. This challenge should be addressed by renaming elements of business process models in order to they faithfully represent the semantics performed actually.

3.5 Ambiguity

Another challenge to be taken into account is ambiguities that may be present in some business process elements. For example, redundancy faults sometimes occur during reverse engineering owing to different source code pieces (e.g., two callable units) lead to build two redundant business tasks that actually are part of a more complex task (e.g., a business task supported by both callable units). Fig. 5 gives an example where there are two different paths are obtained between Task 1 and Task 2 (direct path and conditional path) from different source code pieces. One of the paths contradicts the logic of the other path since the second is more restrictive.

Ambiguity is important to determinate the quality of business process models since it affects the understandability and modifiability of the model, i.e., how far elements in the model are intuitively formulated [22]. The ambiguity, therefore, negatively affects the ability to communicate efficiently the behavior of the business process. A model is considerate unambiguous when it is free of redundancies and it contains no elements that contradict the logic of other element. The ambiguity must be addressed by detecting and removing redundancies and inconsistencies in a business process model.

4 Business Process Model Improvement Approach

In order to address the challenges outlined above, this paper presents an approach for improving business process models obtained from information systems with the aim that they reflect as faithfully as possible the business reality with optimal levels of quality. This approach proposes three stages: repairing, refactoring and expert-based improvement. Each stage addresses some challenges above mentioned and uses some knowledge sources to carry out its purpose. Fig. 6 symbolizes the horseshoe model that characterizes the reengineering, where the upper stages (refactoring and expert-based improvement) represent a higher abstraction level than the bottom stage (repairing).

Repairing stage is considerate in reverse engineering level since it uses knowledge sources such as recorded event logs to address the completeness challenge. The aim of this stage is to ensure that business process models reflect the real execution of the information system. Preliminary results concerning this stage are given in [26].

That work shows a set of steps that are carried out taking as input a business process model and event logs and returning as output an enhanced business process model with additional sequence flows retrieved from event logs. The technique detects unrecovered sequence flows as regards the event log and tidily adds these sequence flows to the target business process model. After the conduction of a case study to demonstrate the feasibility of the technique, the results show that the fitness of the process model increases, i.e., repairing business process model leads to a more faithful representation of the observed behavior.

Refactoring stage is concerning to modify the internal structure of business process models without changing or altering the external behavior. This stage maintains the abstraction level while maintaining the semantic. Refactoring techniques therefore improve the quality of business processes, so that they become more understandable, maintainable and reusable [12]. This stage addresses some challenges as relevancy, granularity, uncertainty and completeness. Guidelines, literature, heuristics and experience are additional resources used in this stage. Some refactoring operators are introduced in [27], especially designed for use with reversed business process models. For example, some refactoring operators address the relevancy by means of the elimination of isolated nodes, unnecessary nesting, among other. Other refactoring operators address the granularity by grouping elements. Other refactoring operators address the completeness following good practices in business process modeling and incorporating additional needed elements. Some results obtained after applying these refactoring operators are shown in next section. Section 5 gives some results of this stage. There, each refactoring operator is applied in isolation in order to visualize the change that each operator provides to the business process model.

Finally, expert-based improvement stage addresses ambiguity and relevancy by means of expert decision. This stage is because not all challenges can be addressed automatically by the previous two stages, it is necessary also the opinion and feedback of an expert in certain situations to improve the business process model.

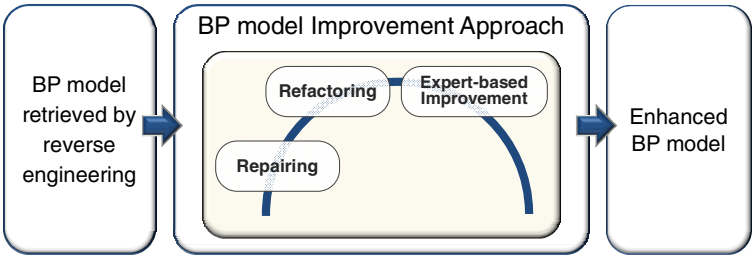


Fig. 6. Proposed improvement approach by means of three stages: repairing, refactoring and expert-based improvement

5 Refactoring Results

This section shows some results obtained in the second stage considered in the approach. In order to illustrate the effect of refactoring operators on business process models some aspects are defined:

Business process models taken as *independent variables* have been mined from the source code using MARBLE, the business process archeology tool used to figure out the above challenges (cf. Section 2). The selected information system was *Tabula*, a web application of 33.3 thousands of lines of code devoted to create, manage and simulate decision tables for associating conditions with domain-specific actions. From this information system was retrieved 15 business process models.

Measures used to assess the understandability and modifiability of a business process model [28] are considered as *dependent variables*: the *size* (number of elements such as tasks, events, gateways and data objects), *density* (ratio between the total number of flows in a business process model and the theoretical maximum number of possible flows regarding the number of elements), and *separability* (the ratio between the number of nodes that serve as bridges between otherwise strongly-connected components and the total number of nodes) of the model.

With the aim to illustrate briefly the result obtained by refactoring stage some refactoring operators are used from [27]: **R1** removes nodes (i.e., tasks, gateways or events) in the business process model that are not connected with any other node in order to contribute to the removing of non-relevant elements; Similarly, **R2** removes elements in the business process model that are considered sheet nodes; **R6** creates compounds tasks grouping several small tasks that support another main task. The goal is to remove the fine-grained granularity; **R7** combines data objects that are used for the same task in order to remove the fine-grained granularity; **R8** joins the start and end event to the starting and ending tasks, respectively, to complete the model; **R9** adds join and split gateways that are not present in branches in an effort to complete the model.

After the application of each refactoring operator on each business process model in isolation, values for each dependent variable are collected in Table 1, as well as the gain obtained with respect to the original value. The gain is defined as the ratio between the difference of measure values and the original measure value. Hence, a positive gain means that the refactoring affects the measure positively while a negative gain means that the refactoring affects the measure negatively. A zero gain means that the value for a certain measure did not change after refactoring.

Table 1. Effect of each refactoring operator on the size, density and separability

	Size		Density		Separability	
	Mean	Gain	Mean	Gain	Mean	Gain
Original	35.200	0.000	0.110	0.000	15.533	0.000
R1	30.667	0.395	0.196	-0.607	11.000	0.460
R2	34.400	0.011	0.113	-0.023	14.733	0.019
R6	33.267	0.059	0.106	0.031	15.667	-0.006
R7	33.667	0.026	0.106	0.009	14.600	-0.003
R8	37.600	-0.410	0.207	-0.338	17.933	-0.447
R9	59.400	-0.142	0.105	0.076	15.600	-0.002

Table 1 reveals that removing isolated nodes decreases the size and separability while the density is increased. Despite the density is higher after R1, the relevance of the model has been increased since non-relevant elements have been removed. Similarly, R2 causes an increase of density when the size is decreased. Separability is decreased slightly. R6 creates compound tasks in several business process models.

This fact entails a decrease in the size and density while separability increases slightly. The same happens with R7, the number of nodes and the density is lower but separability is higher. Nevertheless, all measures after R8 are higher due to business process models were incomplete. R9, in turn, cause a significant increase in the size because there were several incoming and outgoing branches without gateways in the original business process models. The same occur with the separability after R9 while density decreases slightly.

6 Conclusions

Reverse engineering has become in a suitable solution to mine business process model from existing information system. Unfortunately, these retrieved business process models entail some challenges that are necessary to address in order to increase their quality degree.

Completeness is an important challenge to deal with in retrieved business process model since data are distributed in several sources. Different types of granularity are also a challenge to address because fine-granularity causes the degree of quality is lower. Moreover, non-relevant information causes a low degree quality since the model should not contain additional elements that do not carry out any business logic in the organization. The uncertain labeling of elements may negatively affect the understandability and therefore an appropriate convention should be followed. In addition, ambiguity is another challenge because a model should be free of redundancies and inconsistencies.

It is with all the above in mind that this paper presents an approach for improving business process models obtained from information systems in an effort to deal with above challenges. The approach defines three stages: repairing, refactoring and expert-based improvement. These stages address challenges above mentioned by using additional knowledge sources to perform its goal. Moreover, in order to illustrate one of the stages, this work presents some results of refactoring stage. The result shows that the measures selected for assessing the quality of business process models -in terms of their understandability and modifiability, are improved in the most of cases by removing non-relevant and fine-grained elements as well as by completing models. Despite the fact that this work applies refactoring operator in isolation, studies reveal that refactoring operators do not satisfy commutative property among them, making necessary to figure out the best execution order [27].

After the completion of this work a set of future works has been identified: (1) Refining the repairing stage in order to obtain more valuable information from event logs in order to repair retrieved business process models; (2) Refining the refactoring stage by defining new refactoring operators to address more challenges. In addition, the use of more measures for assessing the understandability and modifiability is required; (3) Definition of expert-based improvement stage by means of the use of expert decision to remove ambiguities in the business process model.

Acknowledgements. This work was supported by the FPU Spanish Program and the R&D projects MAGO /PEGASO (Ministerio de Ciencia e Innovación [TIN2009-13718-C02-01]) and GEODAS-BC (Ministerio de Economía y Competitividad & Fondos FEDER [TIN2012-37493-C03-01]).

References

1. Weske, M.: Business Process Management: Concepts, Languages, Architectures, Leipzig, Germany, p. 368. Springer, Heidelberg (2007)
2. Jeston, J., Nelis, J., Davenport, T.: Business Process Management: Practical Guidelines to Successful Implementations, 2nd edn. Butterworth-Heinemann (Elsevier Ltd.), NV (2008)
3. OMG. Business Process Modeling Notation Specification 2.0 (2011), http://www.omg.org/spec/BPMN/2.0/PDF/
4. Pérez-Castillo, R., et al.: MARBLE. A Business Process Archeology Tool. In: 27th IEEE International Conference on Software Maintenance (ICSM 2011), Williamsburg, VI, pp. 578–581 (2011)
5. Fernández-Ropero, M., Pérez-Castillo, R., Piattini, M.: Refactoring Business Process Models: A Systematic Review. In: Filipe, J., Maciaszek, L. (eds.) 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), pp. 140–145. SciTePress, Wrocław (2012)
6. van der Aalst, W.: Process Mining: Overview and Opportunities. ACM Transactions on Management Information Systems (TMIS) 3(2), 7 (2012)
7. Indulska, M., Recker, J., Rosemann, M., Green, P.: Business process modeling: Current issues and future challenges. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 501–514. Springer, Heidelberg (2009)
8. Fahland, D., van der Aalst, W.M.P.: Repairing Process Models to Reflect Reality (2012)
9. Weber, B., Reichert, M.: Refactoring Process Models in Large Process Repositories. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 124–139. Springer, Heidelberg (2008)
10. Dijkman, R., Rosa, M.L., Reijers, H.A.: Managing large collections of business process models—Current techniques and challenges. Computers in Industry 63(2), 91 (2012)
11. Weber, B., et al.: Survey paper: Refactoring large process model repositories. Comput. Ind. 62(5), 467–486 (2011)
12. Dijkman, R., et al.: Identifying refactoring opportunities in process model repositories. Information and Software Technology (2011)
13. La Rosa, M., et al.: Managing process model complexity via abstract syntax modifications. IEEE Transactions on Industrial Informatics 7(4), 614–629 (2011)
14. Uba, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: Clone detection in repositories of business process models. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 248–264. Springer, Heidelberg (2011)
15. Ekanayake, C.C., Dumas, M., García-Bañuelos, L., La Rosa, M., ter Hofstede, A.H.M.: Approximate clone detection in repositories of business process models. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012. LNCS, vol. 7481, pp. 302–318. Springer, Heidelberg (2012)
16. Pittke, F., Leopold, H., Mendling, J.: Spotting Terminology Deficiencies in Process Model Repositories. In: Nurcan, S., Proper, H.A., Soffer, P., Krogstie, J., Schmidt, R., Halpin, T., Bider, I. (eds.) BPMDS 2013 and EMMSAD 2013. LNBIP, vol. 147, pp. 292–307. Springer, Heidelberg (2013)

17. Smirnov, S., Weidlich, M., Mendling, J.: Business process model abstraction based on synthesis from well-structured behavioral profiles. International Journal of Cooperative Information Systems 21(1), 55–83 (2012)
18. Smirnov, S., Reijers, H.A., Weske, M.: A semantic approach for business process model abstraction. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 497–511. Springer, Heidelberg (2011)
19. Polyvyanyy, A., Smirnov, S., Weske, M.: Business process model abstraction. In: Handbook on Business Process Management, vol. 1, pp. 149–166 (2010)
20. Smirnov, S.: Business process model abstraction, Universitätsbibliothek (2012)
21. Pérez-Castillo, R., et al.: A family of case studies on business process mining using MARBLE. Journal of Systems and Software 85(6), 1370–1385 (2012)
22. Overhage, S., Birkmeier, D.Q., Schlauderer, S.: Quality Marks, Metrics, and Measurement Procedures for Business Process Models. Business & Information Systems Engineering, 1–18 (2012)
23. Pérez-Castillo, R., et al.: Generating Event Logs from Non-Process-Aware Systems Enabling Business Process Mining. Enterprise Information System Journal 5(3), 301–335 (2011)
24. Zou, Y., Hung, M.: An Approach for Extracting Workflows from E-Commerce Applications. In: Proceedings of the Fourteenth International Conference on Program Comprehension 2006, pp. 127–136. IEEE Computer Society (2006)
25. Binkley, D., et al.: To camelcase or under_score. IEEE (2009)
26. Fernández-Ropero, M., Reijers, H.A., Pérez-Castillo, R., Piattini, M.: Repairing Business Process Models as Retrieved from Source Code. In: Nurcan, S., Proper, H.A., Soffer, P., Krogstie, J., Schmidt, R., Halpin, T., Bider, I. (eds.) BPMDS 2013 and EMMSAD 2013. LNBIP, vol. 147, pp. 94–108. Springer, Heidelberg (2013)
27. Fernández-Ropero, M., et al.: Assessing the Best-Order for Business Process Model Refactoring. In: 28th Symposium On Applied Computing (SAC), Coimbra, Portugal, pp. 1400–1406 (2013)
28. Fernández-Ropero, M., et al.: Quality-Driven Business Process Refactoring. In: International Conference on Business Information Systems (ICBIS), Paris, France, pp. 960–966 (2012)

Measuring the Effect of Enabling Traces Generation in ATL Model Transformations

Iván Santiago, Juan M. Vara, Valeria de Castro, and Esperanza Marcos

Kybele Research Group, Rey Juan Carlos University,
Avda. Tulipán S/N, 28933 Móstoles, Madrid, Spain
{ivan.santiago,juanmanuel.vara,valeria.decastro,
esperanza.marcos}@urjc.es
http://www.kybele.es

Abstract. The benefits that proper management of traceability information can bring to any given (software development) project are beyond any doubt. These benefits become even more appealing when dealing with traceability does not imply additional efforts. This is the case of Model-Driven Engineering (MDE). As a matter of fact, since model transformations are the wheel that drives MDE proposals forward, traceability data can be automatically available in MDE projects. To that end, the implicit traceability relationships contained in any model transformation have to be made explicit by enriching the model transformation with traces generation capabilities. However, this refinement process implies a cost in terms of quality: enriched transformations are intuitively more complex. To back such intuition, this work presents an empirical study to assess the impact over the quality of the automatic enrichment of model transformations.

Keywords: Model-driven Engineering, Model Transformations, Traceability, Quality Metrics.

1 Introduction

The management of traceability in software development projects implies keeping track of the relationships between the different software artifacts produced along the process. This way, appropriate management of traceability helps to monitor the evolution of system components and carry out different software activities such as change impact assessment, requirements validation, maintenance tasks, etc. [1].

Unfortunately, generating and maintaining links among different software artifacts is a tedious, time-consuming and error prone task if no tooling support is provided to that end [2]. In this sense, the advent of the Model-Driven Engineering (MDE) paradigm, which principles are to enhance the role of models and to increase the level of automation all along the development process [3], provides a new landscape that can positively influence the management of traceability [4]. Indeed, MDE brings new scenarios where appropriate management of traceability is almost mandatory, such as model synchronization or incremental model changes [5], all of them particular scenarios of software evolution.

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 229–240, 2013.

The key to foster automation in MDE projects are the model transformations that connect the different models involved in the proposal [6]. Simply put, a model transformation defines a set of relationships between the elements of source and target metamodels that must hold between the elements of the models conforming to such metamodels [7]. Therefore, a model transformation contains implicit information from which trace-links (traces) can be derived. Actually, such links can be seen as instances of the relationships defined at metamodel-level. Therefore, if we made explicit this information in the model transformation itself, it could generate, apart from the corresponding target models, an *extra* model which contains the traces between the elements of the models involved in the transformation.

Nevertheless, the enrichment of model transformations to support the production of traces model might have an impact over the quality of the transformation. This paper focuses on the assessment of such impact. To that end, it leans on some previous works by van Amstel and van den Brand [8,9] who defined a set of quality metrics for model transformations and tried to relate them with some quality attributes; such as understandability, modifiability, reusability, completeness, consistency and conciseness.

In particular, this work provides an empirical study of the impact of enriching ATL (*Atlas Transformation Language*) [10] model transformations. To that end, an heuristic to obtain quantitative indicator to assess the quality of model transformations is introduced. Such indicator is then used to compare standard and enriched versions of 7 model transformations with different levels of complexity.

The rest of this work is structured as follows: Section 2 describes the enrichment process for ATL model transformations supported by `iTrace` [11]; Section 3 introduces the proposal from van Amstel and van den Brand for the quality assessment of model transformations; Section 4 presents the empirical study performed in this work and the analysis of results; and finally Section 5 concludes by highlighting the main findings and providing directions for further work.

2 Enriching ATL Model Transformations with iTrace

The first step towards the appropriate management of traceability is the existence of traces. However, since many projects do not provide such traces, mechanisms are needed to support the production of traces. In order to avoid accidental complexity, such mechanisms should be completely automatic and transparent for the user.

To fulfill these requirements in the context of MDE, `iTrace` [11] supports the production of trace models in two different scenarios. On the one hand, it supports the enrichment of model transformations that were developed with model transformation languages which do not support the generation of traces. On the other hand, it bundles a set of transformations to normalize existing traces models to a common metamodel: the `iTrace` metamodel. In this paper, only the first scenario is considered, i.e., the production of trace models by enriching existing model transformations. More specifically, we focus on the generation of trace models from enriched ATL transformations.

ATL provides limited access to the target elements generated by running a transformation, e.g. in the current version of the ATL virtual machine (ATL-VM), target elements cannot be selected according to their type. Besides, the ATL-VM discards

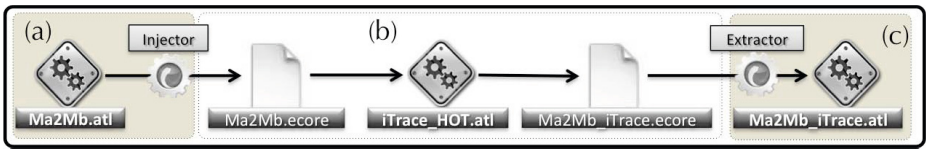


Fig. 1. Adding traceability capabilities in ATL transformations - adapted from [14]

the tracing information after the transformation is run. This implies that ATL model transformations should be refactored to support the production of trace models [12]. However, such refactoring can be automated by using High-Order Transformations (HOT)[13], i.e. *"a model transformation such that its input and/or output models are themselves transformation model"*

This way, HOTs are used to enrich existing m2m transformations so that they are able to produce not only the corresponding target models, but also trace models. This idea was first proposed by Jouault in [14] that introduced an initial prototype to support the enrichment of ATL transformations. The enrichment process bundled in `iTrace` is a little bit more complex than the one from [14], due to the increased complexity of `iTrace` metamodels.

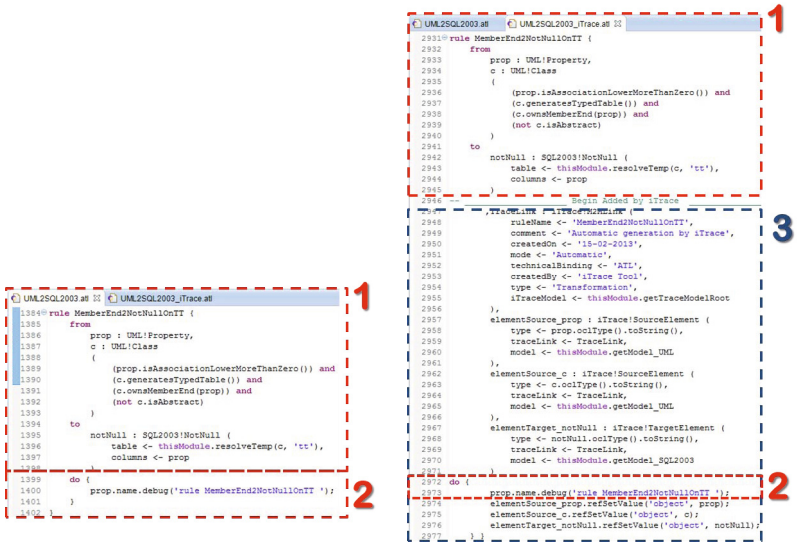
Figure 1 depicts graphically the enrichment process for m2m transformations supported by `iTrace`: first, the TCS [15] injector/extractor for ATL files bundled in the AMMA (Atlas Model Management Architecture) platform[1] produces a transformation model from a given ATL transformation (a); next, such transformation model is enriched by a HOT (b) and finally the resulting transformation models is again serialized into an ATL model transformation (c). As mentioned before, the execution of such enriched transformation will produce not only the corresponding target models, but also a traces model.

The result of this enrichment process is partially illustrated in Figure 2 which shows two excerpts from the original transformation and its enriched version. More concretely, Figure 2(a) shows the original `MemberEnd2NotNullOnTT` mapping rule while Figure 2(b) shows the version of such rule produced by the enrichment process. (1) and (2) denote the statements that are preserved in the enriched version whereas (3) identifies the statements added to support traces generation.

Finally, with regard to the selection of ATL, there were two decisive factors. Firstly, ATL is considered the *de facto* standard for the development of model transformations [16] and it has additionally been developed according to MDE principles. As a result, it provides a complete metamodel that allows ATL model transformations to be modeled without the need to define a new metamodel. Note that the the metrics defined by van Amstel and van den Brand are computed by executing a set of model transformations over the transformation models obtained from the source-code that implements the transformations under study.

However, the evaluation of other transformation languages is technically feasible, since any metamodel-based transformation language facilitates obtaining a

[1] The AMMA Platform. Available in:
`http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/`



(a) Original transformation (b) Enriched transformation

Fig. 2. Enrichment of the `MemberEnd2NotNullOnTT` mapping rule

transformation model from a given transformation. Computing the metrics for such language only requires the adaptation of the set of transformations aforementioned to the metamodel of the targeted language.

3 Quality Metrics for Model Transformations

Despite the crucial role of model transformations in MDE, few works focused on their quality can be found in the literature. Probably the most mature are those from van Amstel and van den Brand.

In [8] the authors propose a set of metrics for ATL model transformations. Such metrics are classified into 4 groups: rule metrics, helper[2] metrics, dependency metrics and miscellaneous. Besides, they introduce the `ATL2Metrics` tool that automates the measurement process for any given (ATL) transformation.

The idea, illustrated in Figure 3, leans also on the use of HOTs: a transformation model is obtained from a given ATL transformation. Next, such model is consumed by the (`Metrics extractor`) transformation to produce a metrics model. Finally, the metrics models is serialized to the `csv` format by a `Pretty printer`.

Next, in [9] van Amstel and van den Brand lean on the previously defined metrics to develop a proposal to assess the quality of model transformations. The idea was to identify the relationships between the metrics and a set of quality attributes. In the following, adapted definitions[3] from those published in [18] are provided for such attributes:

[2] ATL helpers can be viewed as the ATL equivalent to methods. They make it possible to define factorized ATL code that can be called from different points of an ATL transformation.

[3] Other definitions can be found in [17], pp 10.

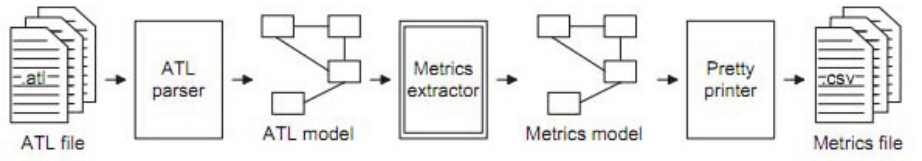


Fig. 3. `ATL2metrics` architecture (from [8])

Conciseness — A transformation possesses the characteristic conciseness to the extent that excessive information is not present. This implies that the transformation is not excessively fragmented into modules, overlays, functions and subroutines, nor that the same transformation is repeated in numerous places, rather than defining a subroutine or macro; etc.

Consistency — A transformation possesses the characteristic *internal consistency*[4] to the extent that it uses a uniform notation, terminology and symbology within itself, and *external consistency* to the extent that the content is traceable to the requirements.

Completeness — A transformation possesses the characteristic completeness to the extent that all its parts or components are present and each part is fully developed. For instance, a mapping rule possesses the characteristic completeness to the extent that it does not need from external invocations to produce target elements.

Modifiability — A transformation possesses the characteristic modifiability to the extent that it facilitates the incorporation of changes, once the nature of the desired change has been determined. Note the high level of abstraction of this characteristic in contrast with that of augmentability[5].

Reusability — A transformation possesses the characteristic *reusability* to the extent that it can be operated easily and well on metamodels other than its current ones.

Understandability — A transformation possesses the characteristic *understandability* to the extent that its purpose is clear to the inspector. This implies that variable names or symbols are used consistently, modules of code are self-descriptive, and the control structure is simple or in accordance with a prescribed standard, etc.

To that end, a poll on the quality of a given set of transformations was conducted between ATL experts. To establish the relations between their observations and the data gathered running the metrics, the Kendall correlation test was used. This test returns two values, viz. a correlation coefficient (`cc`) and a significance value (`sig`). The correlation coefficient indicates the strength and direction of the correlation. A positive correlation coefficient means that there is a positive relation between the metric and the

[4] Internal consistency implies that coding standards are homogeneously adhered to; e.g., comments should not be unnecessarily extensive or wordy at one place, and insufficiently informative at another, that number of arguments in subroutine calls match with subroutine header, etc.

[5] `Augmentability`: A transformation possesses the characteristic *augmentability* to the extent that it can easily accommodate expansion in component computational functions. This is a necessary characteristic for modifiability.

Table 1. Kendall's correlations (from [9])

Metric	Comple.		Modifi.		Consis.		Reusab.		Concis.		Unders.	
	cc	*sig*	*cc*	*sig*	*cc*	*sig*	*cc*	*sig*	*cc*	*sig*	*cc*	*sig*
# Elements per output pattern	-.228	.180	-.215	.202	.124	.472	-.146	.389	-.122	.474	-.375	.026
# Calls to resolveTemp()	-.159	.380	-.358	.045	-.088	.632	-.236	.189	-.179	.323	-.352	.049
# Calls to resolveTemp per rule	-.106	.558	-.306	.087	-.061	.741	-.236	.189	-.153	.399	-.326	.068
# Parameters per called rule	-.391	.038	-.407	.029	-.122	.524	-.345	.066	-.218	.249	-.407	.029
# Unused parameters per called rule	-.391	.038	-.407	.029	-.122	.524	-.345	.066	-.218	.249	-.407	.029
Called rule fan-in	-.391	.038	-.407	.029	-.122	.524	-.345	.066	-.218	.249	-.407	.029
Unit fan-in	-.391	.038	-.407	.029	-.122	.524	-.345	.066	-.218	.249	-.407	.029
Unit fan-out	-.391	.038	-.407	.029	-.122	.524	-.345	.066	-.218	.249	-.407	.029
# Input models	-.391	.038	-.407	.029	-.122	.524	-.345	.066	-.218	.249	-.407	.029
# Ouput models	-.391	.038	-.407	.029	-.122	.524	-.345	.066	-.218	.249	-.407	.029
# Units	-.391	.038	-.407	.029	-.122	.524	-.345	.066	-.218	.249	-.407	.029
# Unused helpers	-.391	.038	-.407	.029	-.122	.524	-.345	.066	-.218	.249	-.407	.029
# Times a unit is imported	-.379	.045	-.138	.459	.086	.654	-.084	.656	-.247	.192	-.318	.088
Lazy rule fan-in	-.397	.021	-.143	.404	.005	.976	-.021	.905	-.062	.719	-.356	.037
Helper cyclomatic complexity	-.357	.037	-.154	.364	-.026	.882	-.175	.304	.126	.461	-.248	.142
# Direct copies	.322	.070	.040	.822	-.059	.745	-.125	.478	-.196	.271	.227	.197
# Imported units	-.323	.078	-.080	.661	.110	.554	-.027	.883	-.223	.225	-.252	.164
Rule fan-out	-.333	.046	-.124	.453	-.157	.351	-.235	.157	.024	.885	-.223	.175
Helper fan-out	-.105	.537	.183	.278	.302	.081	.264	.120	.136	.426	.020	.907
# Transformation rules	-.082	.623	-.086	.604	-.364	.031	-.273	.099	-.092	.582	.024	.885
# Called rules	-.158	.389	-.263	.149	-.308	.098	-.365	.046	-.347	.060	-.128	.482
# Unused called rules	-.158	.389	-.263	.149	-.308	.098	-.365	.046	-.347	.060	-.128	.482
# Rules with filter	-.005	.977	-.038	.818	-.049	.771	-.129	.435	-.402	.016	-.005	.977
# Rules with local variables	.032	.861	-.051	.780	-.137	.460	-.178	.328	.302	.100	-.013	.944
# Rules per input pattern	-.130	.434	-.114	.489	.029	.861	-.014	.931	.315	.059	-.109	.507
# Unused input pattern elements	-.033	.854	.059	.737	-.056	.758	.033	.854	.297	.097	-.032	.854
# Variables per helper	.013	.944	.063	.727	-.111	.550	.178	.328	.328	.074	.006	.972
# Non-lazy matched rules	.034	.839	.000	1.000	-.029	.861	-.129	.435	-.383	.022	.014	.931
# Helpers per helper name (overloadings)	-.054	.769	-.066	.714	.220	.237	.060	.741	.007	.971	-.033	.855
# Variables per rule	.070	.700	-.076	.676	-.111	.550	-.242	.184	.225	.220	-.038	.834
Helper fan-in	-.024	.885	.000	1.000	-.236	.162	-.196	.236	-.005	.977	.138	.402
# Helpers	-.201	.243	-.229	.180	-.047	.786	-.246	.152	.187	.281	-.178	.296
# Unused lazy matched rules	-.152	.419	.138	.460	.026	.892	.076	.687	.217	.251	-.025	.893
# Rules with do-section	-.057	.747	-.153	.385	.018	.922	-.108	.540	-.173	.332	.000	1.000
# Lazy matched rules	-.201	.243	.041	.812	-.058	.741	.051	.765	.239	.168	-.081	.633
# Helpers per unit	-.201	.243	-.229	.180	-.047	.786	-.246	.152	.187	.281	-.178	.296

quality attribute and a negative correlation coefficient implies a negative relation. The significance indicates the probability that there is no correlation between the metric and the quality attribute even though one is reported, i.e., the probability for a coincidence. Note that correlation does not indicate a causal relation between the metric and the quality attribute.

Figure 1 shows the correlations that were identified in the study of van Amstel and van den Brand.

4 Evaluation

In order to improve the rigor of this study, we have followed the guidelines for conducting case studies proposed by Runeson and Höst in [19]. In particular, we have adapted the protocol used in [20] which is based on the proposal of Runeson and Höst. In essence, the adapted protocol distinguishes a set of stages, namely: case selection, design and execution, data collection and finally analysis and interpretation. The highlights of each stage are presented as follows.

Table 2. ATL transformations selected

ID	Transformation	Purpose	LOC	MR	IN/OUT
T1	ASD2WSDL	Maps Abstract Service Descriptions (ASD) into WSDL models.	236	13	1/1
T2	Class2Relational	Maps UML class diagrams into relational models.	112	6	1/1
T3	Families2Persons	Maps Families models into People models.	46	2	1/1
T4	SQL20032ORDB4ORA	Maps ORDB models that conform to the SQL:2003 standard into ORDB models for Oracle.	1247	51	1/1
T5	UML2SQL2003	Maps UML class diagrams annotated by means of a AMW (*Atlas Model Weaver*) models into ORDB models that conform to the SQL:2003 standard.	2181	66	2/1
T6	UML2XMLSchema	Maps UML class diagrams annotated by means of a AMW models into XSD models.	459	13	2/1
T7	WSDL2ASD	Maps WSDL models into ASD models.	190	9	1/1
	TOTAL		**4471**	**160**	**9/7**

4.1 Case Studies Selection

In order to consider different sizes and levels of complexity, 7 case studies were selected. Their main features are summarized in Table 2. From left to right, the following information is provided for each transformation: identifier (`ID`); name (`Transformation`); functionality delivered (`Purpose`); # of lines of code (`LOC`); # of mapping rules (`MR`); # of source and target models (`IN/OUT`).

4.2 Design and Execution

The empirical study has been executed as follows:

(1) Original transformation is checked and run to collect the # of LOC and execution time.
(2) `ATL2Metrics` is run over the transformation to gather values for each metric.
(3) Transformation is automatically enriched using the `iTrace` framework to support the production of trace models. The enrichment process is process was described in Section 2.
(4) Enriched transformation is checked and run to collect the # of LOC and execution time.
(5) `ATL2Metrics` is run over enriched transformation to gather values for each metric.
(6) Steps 1 to 5 are repeated for each transformation under study.
(7) Data collected is analyzed.

The values gathered are then computed to obtain an overall indicator of the quality of each model transformation. This computation is supported by the following heuristic[6] that exploits somehow the data provided by van Amstel and van den Brand.

[6] Authors would like to thank Dr. Diego Vidaurre, from the Oxford center for Human Brain Activity, for his valuable advice on the statistical analysis of the results.

Let n be the number of metrics, p the number of attributes and k the number of transformations whose attributes we aim to estimate. Let $X \in [-1,-1]^{n \times p}$ be the matrix containing the Kendall correlation coefficients for each pair of metric and attribute. Let $Y \in \mathbb{R}^{n \times k}$ be the matrix containing the metrics for each transformation. The objective is to estimate a matrix $\tilde{Z} \in [0,-1]^{p \times k}$ with the attributes for each transformation.

Then, as Equation 1 shows we can compute Z just as the weighted average of the corresponding metrics, where the weights are given by the correlation coefficients. Finally, Equation 2 scales $\tilde{Z}$ so that each element ranges from 0 to 1.

$$Z_{jl} = \frac{\sum_{i=1}^{n} Y_{il} \cdot X_{ij}}{\sum_{i=1}^{n} |X_{ij}|} \tag{1}$$

$$\tilde{Z}_{jl} = \frac{Z_{jl} - min\,(Y_{\cdot l})}{max\,(Y_{\cdot l}) - min\,(Y_{\cdot l})} \tag{2}$$

4.3 Data Collection

Table 3 summarizes the results obtained from the execution of the previous process. First column identifies the transformation under consideration, where `Tx` stands for the original version of the transformation and `Tx'` for the enriched one.

Then the value for each quality attribute, as well as the overall value (arithmetic average) for each transformation are shown. Note that an additional value is shown in those rows corresponding to enriched transformations. It states the difference between the values obtained by the original and enriched versions of the transformations. For instance, the understability value for `T1` is 0.93 whereas the one for `T1'` is 0.75. Understability of `T1` has consequently decreased 18.43% because of the enrichment process.

Last row sums up the data by showing the average values and differences of each attribute and the overall indicator.

4.4 Analysis and Interpretation

To ease the analysis and interpretation of the data collected, we first introduce the main findings to later dig into the data collected regarding each quality attribute.

General Overview. According to Table 3, the enrichment of model transformations to support the production of trace models has a negative impact over the quality of the transformations. On average such impact is about 20%. This negative influence becomes more pronounced as the quality of the original transformation gets worse. See for instance the impact of enrichment over the quality of `T5`.

As a matter of fact, these evidences are aligned with the initial intuition since the enrichment of the transformations implies adding extra LOC to implement the machinery that will generate the traces. Besides, bigger transformations are more affected: the more mapping rules the original transformation contains, the more machinery have to be added in the enriched version of the transformation.

Table 3. Data collection overview

Transf.	Comple.	Modifi.	Consis.	Reusab.	Concis.	Unders.	Quality
T1	**0.96**	**1.00**	**0.88**	**0.88**	**0.96**	**0.93**	**0.93**
T1'	**0.60** *-36.64%*	**0.90** *-9.89%*	**0.71** *-17.04%*	**0.70** *-18.10%*	**0.71** *-24.47%*	**0.75** *-18.43%*	**0.73** *-20.76%*
T2	**1.00**	**1.00**	**0.96**	**0.96**	**0.97**	**0.97**	**0.98**
T2'	**0.69** *-30.95%*	**0.90** *-9.89%*	**0.78** *-17.04%*	**0.78** *-18.10%*	**0.73** *-24.47%*	**0.80** *-16.79%*	**0.78** *-19.54%*
T3	**0.84**	**1.00**	**1.00**	**1.00**	**0.99**	**1.00**	**0.97**
T3'	**0.65** *-19.05%*	**0.90** *-9.89%*	**0.83** *-17.04%*	**0.82** *-18.10%*	**0.75** *-24.47%*	**0.88** *-12.36%*	**0.80** *-16.82%*
T4	**0.55**	**1.00**	**0.38**	**0.40**	**0.66**	**0.93**	**0.65**
T4'	**0.34** *-21.13%*	**0.91** *-8.90%*	**0.21** *-17.04%*	**0.22** *-17.95%*	**0.40** *-25.85%*	**0.71** *-21.26%*	**0.47** *-18.69%*
T5	**0.71**	**0.09**	**0.22**	**0.22**	**0.31**	**0.22**	**0.29**
T5'	**0.33** *-37.18%*	**0.00** *-9.44%*	**0.00** *-21.58%*	**0.00** *-22.47%*	**0.00** *-30.59%*	**0.00** *-22.10%*	**0.06** *-23.89%*
T6	**0.32**	**0.37**	**0.89**	**0.85**	**0.90**	**0.47**	**0.63**
T6'	**0.00** *-32.02%*	**0.27** *-9.82%*	**0.67** *-21.73%*	**0.62** *-22.60%*	**0.59** *-30.60%*	**0.26** *-20.29%*	**0.40** *-22.84%*
T7	**1.00**	**1.00**	**0.92**	**0.92**	**1.00**	**0.97**	**0.97**
T7'	**0.64** *-35.95%*	**0.90** *-9.89%*	**0.75** *-17.04%*	**0.74** *-18.10%*	**0.76** *-24.47%*	**0.80** *-16.67%*	**0.76** *-20.35%*
Average	**0.62** *-30.42%*	**0.73** *-9.68%*	**0.66** *-18.36%*	**0.65** *-19.34%*	**0.69** *-26.42%*	**0.69** *-18.27%*	**0.67** *-20.41%*

Completeness. The quality attribute most negatively affected by the enrichment process is completeness (30.42% on average). According to the Kendall's coefficient table (see Table 1), it shares some metrics with the rest of quality attributes. However, the completeness values obtained for the different transformations do not follow the same trend than those of the rest of attributes. Therefore, we may conclude that completeness values are mainly derived from the `Helper cyclomatic complexity`, `# Direct copies`, `# Imported units` and `Rule fan-out` metrics since they are related just to the completeness attribute. Indeed, the negative impact could be granted almost exclusively to the `Rule fan-out` metric[7].

In the enriched transformations produced by `iTrace` , an auxiliary mapping rule is called for each element in the source and target pattern of every mapping rule. As a result, complete mapping rules (those able to produce target elements without calling other rules or helpers) are turned into non-complete mapping rules in the enriched version of the transformation, with the consequent impact over completeness of the transformation.

Modifiability. In contrast with the impact on completeness, modifiability is the quality attribute least affected by the enrichment process (9.68% on average). The value of this attribute is mainly conditioned by the number of units and source and target models involved in the transformation. As well, the use of constructions that raise the level of coupling, like the `resolveTemp`[8] operation has a negative impact on modifiability.

As a matter of fact, the enriched transformations produced by `iTrace` do not imply the addition of building-blocks that raise the level of coupling or new units. Besides, the data show that the impact on modifiability does not depend on the size of the original transformation.

[7] The `Rule fan-out` metric computes the average number of external invocations, e.g. a mapping rule invokes a `helper` or another mapping rule.

[8] Allows pointing, from an ATL rule, to any of the target model elements that will be generated. Its use goes against the declarative nature of ATL transformations.

Consistency, Reusability and Conciseness. Unfortunately, the conclusions regarding consistency, reusability and conciseness are not conclusive. This is mainly due to the metrics proposed by van Amstel and van den Brand. Back to the table, transformations can be grouped into two categories attending to the values for these attributes. First group comprises `T1`, `T2`, `T3`, `T4` and `T7`. Second group comprises `T5` and `T6` transformations. If we check which are the features shared by each group, we find out that transformations in the first group involve two source models, while those in the second group involve just one source model.

Nevertheless, Table 1 shows that, in theory, there is no relation between the metric computing the number of source models `# Input models` and the values for three attributes. In order to explain this paradox, we focus on the metrics shared by the attributes under study in this section, namely `# Called rules` and `# Unused called rules`.

The enrichment process supported by `iTrace` results in the addition of a called rule for each source model. Such rule is in charge of linking each source element with its corresponding model in the traces model produced. The more source models involved in the original transformation, the more called rules added in the enriched version. The # of source models has consequently a direct influence over the value delivered by the `# Called rules` metric. All this given we can conclude that, according to the metrics proposed by van Amstel and van den Brand, the # of source models has a direct influence on the consistency, reusability and conciseness of an ATL (enriched) transformation

Understandability. Understability is the attribute for which more scattered values are obtained for enriched transformations, even though it shares a good number of metrics with consistency, reusability and conciseness, for which a common trend was found. Therefore, we focus on the `# Elements per output pattern`[9] metric, since it is the only metric solely related with understability.

Given a mapping rule containing n elements in the source an target patterns, the enriched version of such rule contains $n + 1$ *additional* elements in the target pattern. These additional elements serve to generate n references to the source and target elements related by the rule, plus a trace link element that connect them. The addition of these elements contributes obviously to increment the `# Elements per output pattern` with the consequent impact on understability.

Under the light of these observations, it becomes clear that the disparity in the values for understability comes from the disparity on the values returned by the `# Elements per output pattern` for the original transformations.

5 Conclusions

In order to reason about the cost of having traceability data in MDE projects, this work has presented an empirical study to assess the impact of enriching model transformations over their quality. In particular, the quality of 7 ATL [10] model transformations owning different levels of complexity was assessed before and after the enrichment process.

[9] Average # of elements per output pattern.

To do so, an heuristic has been defined that combines the data provided by battery of metrics related with a set of quality attributes [9]. The heuristic provides a measure for each quality attribute as well as an overall quality measure. Finally, the values gathered have been compared and analyzed.

Probably, the main contribution of this paper is to provide empirical evidence to confirm the intuitive knowledge about the impact of adding trace generation support to model transformations. The data collected show that the quality of enriched versions is worse than that of original transformations (the loss rate is about 20%). This impact has a direct consequence over the effort needed for the maintenance of enriched model transformations.

In order to alleviate this impact we advocate in favor of using model-based techniques. To do so, we must adopt the approach introduced by Bèzivin *et al.* to deal with model transformations as transformations models [21]. From there on, model transformations can be used to handle and produce transformation models. As a matter of fact, this work has partially shown that this approach can be effectively adopted. The completely automated enrichment process supported by `iTrace` is largely based on the use of transformation models.

To conclude, we would like to introduce two considerations about the validity of the study. On the one hand, the results might be partially biased by the nature of the particular enrichment process adopted. The trace models generated by the enriched transformations produced by `iTrace` conform to a particular traces metamodel that was defined as part of the proposal. If a different (traces) metamodel is used, the refinements over the original transformation might be different, as well as the results delivered by the metrics when computed over the enriched version of the transformation.

This drives us to the main threat to validity: the metrics proposed by van Amstel and van den Brand [8,9] and their relation with the quality attributes. To address this issue we are reviewing the metrics in order to add new metrics, as well as refine and eliminate some others. Besides, we plan to carry out a new survey in order to have data to apply a mathematically solid regression methodology to correlate metrics and quality attributes.

Acknowledgements. This research is partially funded by the MASAI project, financed by the Spanish Ministry of Science and Technology (Ref. TIN2011-22617).

References

1. Aizenbud-Reshef, N., Nolan, B., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Systems Journal 45, 515–526 (2006)
2. Mäder, P., Gotel, O., Philippow, I.: Enabling automated traceability maintenance through the upkeep of traceability relations. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) ECMDA-FA 2009. LNCS, vol. 5562, pp. 174–189. Springer, Heidelberg (2009)
3. Schmidt, D.: Model-Driven Engineering. IEEE Computer 39, 25–31 (2006)
4. Santiago, I., Jiménez, A., Vara, J.M., De Castro, V., Bollati, V., Marcos, E.: Model-Driven Engineering As a New Landscape For Traceability Management: A Systematic Review. Information and Software Technology 54, 1340–1356 (2012)
5. Selic, B.: What will it take? A view on adoption of model-based methods in practice. Software and Systems Modeling 11, 513–526 (2012)

6. Bèzivin, J.: In search of a basic principle for model driven engineering. UPGRADE, European Journal for the Informatics Professional 5, 21–24 (2004)
7. Sendall, S., Kozaczynski, W.: Model transformation: The heart and soul of model-driven software development. IEEE Software 20, 42–45 (2003)
8. van Amstel, M.F., van den Brand, M.G.: Quality assessment of ATL model transformations using metrics. In: 3rd International Workshop on Model Transformation with ATL (MtATL 2010), vol. 711, pp. 19–33 (2010)
9. van Amstel, M.F., van den Brand, M.G.: Using metrics for assessing the quality of ATL model transformations. In: 4st International Workshop on Model Transformation with ATL (MtATL 2011), vol. 742, pp. 20–34 (2011)
10. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A Model Transformation Tool. Science of Computer Programming 72, 31–39 (2008)
11. Santiago, I., Vara, J.M., de Castro, M.V., Marcos, E.: Towards the effective use of traceability in Model-Driven Engineering projects. In: Ng, W., Storey, V.C., Trujillo, J.C. (eds.) ER 2013. LNCS, vol. 8217, pp. 429–437. Springer, Heidelberg (2013)
12. Yie, A., Wagelaar, D.: Advanced Traceability for ATL. In: 1st International Workshop on Model Transformation with ATL (MtATL 2009), Nantes, France, pp. 78–87 (2009)
13. Tisi, M., Cabot, J., Jouault, F.: Improving higher-order transformations support in ATL. In: Tratt, L., Gogolla, M. (eds.) ICMT 2010. LNCS, vol. 6142, pp. 215–229. Springer, Heidelberg (2010)
14. Jouault, F.: Loosely coupled traceability for ATL. In: 1st European Conference on Model-Driven Architecture: Traceability Workshop (ECMDA 2005), Nuremberg, Germany, vol. 91, pp. 29–37 (2005)
15. Jouault, F., Bézivin, J., Kurtev, I.: TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In: 5th International Conference on Generative Programming and Component Engineering, GPCE 2006, pp. 249–254. ACM, New York (2006)
16. Vara, J.M., Marcos, E.: A framework for model-driven development of information systems: Technical decisions and lessons learned. Journal of Systems and Software 85, 2368–2384 (2012)
17. van Amstel, M.F., Lange, D.F.J., van den Brand, M.G.: Evaluating the quality of ASF+SDF model transformations. Technical report, Eindhoven University of Technology, Eindhoven, The Netherlands (2009)
18. Boehm, B.W., Brown, J.R., Lipow, M.: Quantitative evaluation of software quality. In: Proceedings of the 2nd International Conference on Software Engineering, ICSE 1976, pp. 592–605. IEEE Computer Society Press, Los Alamitos (1976)
19. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14, 131–164 (2009)
20. Pérez-Castillo, R., de Guzmán, I.G.R., Piattini, M.: Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. Computer Standards & Interfaces 33, 519–532 (2011)
21. Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., Lindow, A.: Model Transformations? Transformation Models! In: Wang, J., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 440–453. Springer, Heidelberg (2006)

Reverse Engineering Applied to CMS-Based Web Applications Coded in PHP: A Proposal of Migration

Feliu Trias, Valeria de Castro, Marcos López-Sanz, and Esperanza Marcos

Kybele Research Group, Rey Juan Carlos University,
C/Tulipan, s/n. 28933 Móstoles, Madrid, Spain
{feliu.trias,valeria.decastro,marcos.lopez,
esperanza.marcos}@urjc.es

Abstract. Increasingly, organizations experience the necessity of migrating their legacy Web applications to new platforms which meet better their needs. For these reasons, these organizations demand reengineering processes that enable this migration in an automatic and standardized way minimizing costs. In the last years, Architecture-Driven Modernization (ADM) has acquired great relevance since it solves most of the problems of traditional reengineering. This is specially crucial in the reengineering of CMS-based Web applications. At time of writing there are no methods that could be used in that context. Hence, we defined an ADM-based method for migrating this kind of Web applications composed of three phases: reverse engineering, restructuring and forward engineering. This method is the framework of the work presented in this paper which is focused on its reverse engineering phase defined by three tasks: 1) knowledge extraction, 2) generation of KDM models and 3) generation of the CMS model. In this paper we explain the implementation of these tasks defining text-to-model (T2M) transformations implemented by a model extractor and model-to-model (M2M) transformations defined in ATL. We use a real example of a CMS-based Web application coded in PHP to show the feasibility of the approach.

Keywords: Content Management System, Web Application, Architecture-driven Modernization, Reverse Engineering and Model-driven Engineering.

1 Introduction

In the last years, the volume of digital content managed by organizations has increased dramatically. To solve it, organizations have based their Web applications on Content Management Systems (CMS) since they are platforms which allow users to collect, manage and publish content in a robust and reliable manner [1].

The CMS market is constantly evolving and organizations experiment the necessity of migrating their CMS-based Web applications to another CMS or to a new version of the same CMS because their current one has become obsolete and it does not meet their needs. Therefore, they find necessary to start a standardized an automatic process of reengineering entailing low risks and costs [2].

J. Filipe and L.A. Maciaszek (Eds.): ENASE 2013, CCIS 417, pp. 241–256, 2013.

Currently, the most successful initiative to standardize and automate the reengineering process is the Architecture-Driven Modernization (ADM) proposed by OMG. ADM advocates for the application of model-driven principles to formalize the reengineering process. It provides several benefits such as reducing development and maintenance costs and extending the life cycle of the legacy systems. ADM develops seven standard metamodels to represent the information involved in a software reengineering process, but currently only three of them are available: Abstract Syntax Tree Metamodel (ASTM) [3], Knowledge Discovery Metamodel (KDM) [4] and Structured Metrics Metamodel (SMM) [5]. These metamodels allow developers to manage software reengineering processes in an integrated and standardized manner as well as saving them time and effort creating their own metamodels [7]. ASTM allows to represent in models (ASTM models) the information about existing software assets of a legacy system in the form of Abstract Syntax Trees (AST) [6]. Otherwise, KDM allows to define models (KDM models) at a higher abstraction level representing semantic information about a software system.

In this context of ADM, we conducted a systematic literature review [7] to assess the state of the art of existing reengineering methods related to CMS-based Web applications. One of the main conclusions derived from such review was that it is necessary to have methods for migrating CMS-based Web applications, but currently there are no one based in ADM making it possible.

Accordingly, we defined an ADM-based method for migrating this kind of Web applications [8]. This ADM-based method comprises the three phases belonging to the reengineering processes [2]: 1) reverse engineering, 2) restructuring and 3) forward engineering. Up to now, our method is focused on open-source CMS such as Drupal [9], Joomla! [10] or Wordpress [11] which are implemented in PHP. We focus on the open-source CMS because of their features, their spread use and relevant acceptance in the market [12].

The work presented in this paper is framed in the ADM-based method we defined. Concretely, we focus on the implementation of its reverse engineering phase which is composed of three tasks: 1) *Knowledge extraction*, in this task ASTM models are extracted from the PHP code by text-to-model (T2M) transformations implemented by a model extractor. For implementation of this model extractor we use Xtext [13]; 2) *Generation of KDM models*, KDM models are generated from the previous ASTM models by means of M2M transformations and 3) *Generation of the CMS model*, from the KDM models and by M2M transformations we generate the CMS Model which conforms to the CMS Common Metamodel [14], the cornerstone of our ADM-based method. For the implementation of these M2M transformations we use ATL [15].

To show the feasibility of this approach we use a real example based on a CMS-based Web application based on Drupal and implemented in PHP.

The rest of this paper is organized as follows: Section 2 explains the research context which frames the work presented in this paper. Section 3 presents the implementation of the reverse engineering phase of our ADM-based method. Section 4 presents the related works and makes a comparison of our ADM-based method with them and, finally, Section 5 presents the conclusions and future works.

2 Research Context

2.1 Architecture Driven Modernization

In recent years, Model-Driven Development (MDD) has proven its usefulness as top-down software development paradigm, and now it is expanded to software reengineering and migration processes [16].

In 2003, OMG proposed the Architecture-Driven Modernization (ADM) initiative which follows the MDD principles. ADM adopts the existing Model-Driven Architecture (MDA) [17] standard which proposes the use of models at different abstraction levels: Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM); and the use of a set of automatic transformations to reach a target system starting from a legacy system following a "horseshoe" process. ADM defines seven standard-based metamodels, but currently only three of them are available: Abstract Syntax Tree Metamodel (ASTM), Knowledge Discovery Metamodel (KDM) and Software Metrics Metamodel (SMM). Our method considers models conforming to ASTM and KDM.

ASTM [3] is the metamodel which represents a low-level view of the system. It allows the representation of the source code's syntax of the legacy system in the form of Abstract Syntax Trees (AST). On the other hand, KDM [4] lets you represent semantic information about a software system, ranging from source code to higher-level abstraction such as GUI events, platforms or business rules. It provides a common interchange format intended to represent existing software assets, allowing tool interoperability at PIM level. KDM comprises several packages (e.g. core, kdm, source, code or action) which are grouped in four layers to improve modularity and separation of concerns (infrastructure, program elements, runtime resource and abstractions).

2.2 CMS Common Metamodel

The context of our work is focused on CMS-based Web applications. This kind of Web applications are considered to present a set of specific features that differ from traditional Web application such as [18]: 1) the dynamic creation of content, content is created and added dynamically by non-technical users of the Web, without requiring the intervention of the webmaster, 2) separation between content and design, the page graphical design is stored in a template and the content is stored in a database or a separate document. To update the content of the Web application is not necessary to master HTML and 3) functionality extension, integration and extension in a CMS-based Web application is among their most valuable features, achieved through module addition. This ensures quick content deployment and provides means for flexible extension, besides promoting efficiency and reducing development costs.

For this reason, in [14] we present the CMS Common Metamodel which defines the key concepts for modeling CMS-based Web applications. This CMS Common Metamodel captures elements such as, theme, vocabulary or module, and other specific elements of the CMS domain. These elements are classified in five different

concerns: 1) navigation, which considers the elements that define the navigational structure of the Web application, 2) presentation, which defines the structure and look-and-feel of the Web pages, 3) content, that captures the data and data type of the information managed by the CMS-based web application, 4) user, defines the elements related to roles of the users and their permissions. 5) CMS Behavior, this concern contains the elements that allow the definition of the different functions performed by the CMS-based Web application.

2.3 ADM-Based Method for Migrating CMS-Based Web Applications

In this section we outline the ADM-based method which we proposed for migrating CMS-based Web applications. As we can see in Fig 1, this method is composed of three phases defining a horseshoe process: a) reverse engineering phase, b) restructuring phase and c) forward engineering phase. In the following, we explain those phases and the tasks involved in them.

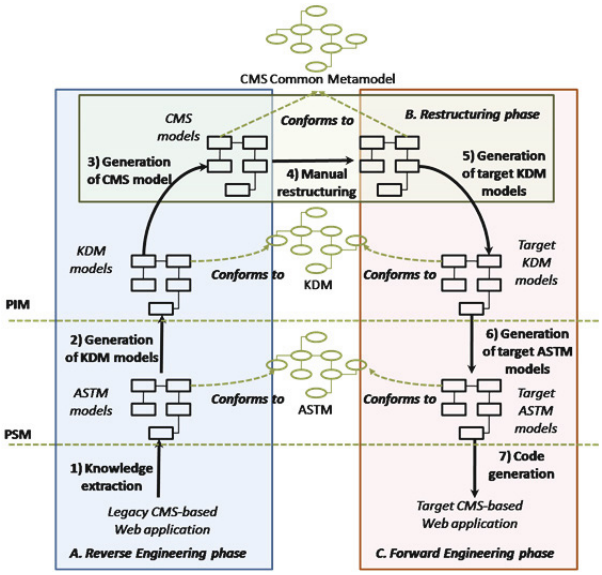


Fig. 1. ADM-based method

A. Reverse Engineering Phase, this is the phase on which we focus in this work. It is composed of three tasks**: 1) Knowledge extraction**, the extraction of ASTM models at PSM level from PHP code by means of a parser (so-called *model extractor*) whose implementation is presented in this paper. ASTM models reduce the complexity to generate the KDM models which are at higher abstraction level; **2) Generation of KDM models**, from those ASTM models we generate automatically KDM models at PIM level by means of M2M transformations. The two KDM models we generate are: Code Model and Inventory Model [19]. It is worth noting that we obtain the Inventory

Model from the legacy code by using MoDisco tool [20], **3) Generation of the CMS Model**, from the KDM models we generate automatically the CMS Model at PIM level also by means of M2M transformations. It conforms to the CMS Common Metamodel and allows the representation of the extracted information within the CMS domain. This metamodel is considered the cornerstone of our ADM-based method.

B. Restructuring Phase, this phase is composed of the **4)** ***Manual restructuring***, this task is in charge of restructuring manually the CMS model extracted from the KDM models into another adapted CMS model at the same abstraction level. In this phase the developer can define new objects or can restructure the inherited ones taking into account the features of the target CMS-based Web application

C. Forward Engineering Phase, this phase defines the top-down development process composed of three tasks: **5)** ***Generation of target KDM models***, from the restructured CMS Model we generate the target Code Model and the target Inventory Model (which conform to KDM) that represent the implementation of the target CMS-based Web application at PIM level; **6)** ***Generation of target ASTM models***, we generate the target ASTM model at PSM level from the target Code Model and the target Inventory Model and **7)** ***Code generation***, we generate the software artifacts that compose the architecture of the target CMS-based Web application (folders and file skeletons) and the code implementing them.

3 Reverse Engineering Phase

In this section we present the implementation of the reverse engineering phase of our ADM-based method which is the focus of the work presented in this paper. As we outlined in Section 2, this reverse engineering phase is composed of three tasks: 1) Knowledge extraction, 2) Generation of KDM models and 3) Generation of the CMS model.

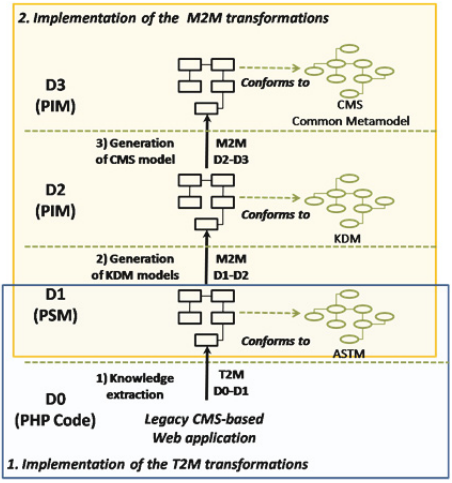


Fig. 2. Reverse engineering phase of our ADM-based method

As we can see in Fig 2, it is possible to classify the reverse engineering phase in four different dimensions: **Dimension 0: (D0)** represents the software artifacts that compose the legacy CMS-based Web application (e.g. source code, database, etc.), in this case it represents PHP code; **Dimension 1: (D1)** represents the ASTM models extracted from the PHP code; **Dimension 2: (D2)** represents the KDM models and **Dimension 3: (D3)** represents the model conforming to the CMS Common Metamodel.

Moreover, the implementation of the reverse engineering phase can be divided in two groups: 1) ***The implementation of the T2M transformations,*** which are the transformations established between D0 and D1 (**T2M D0-D1**) and implements the *Knowledge extraction task* and 2) ***The implementation of the M2M transformations***, which includes the transformations between D1 and D2 (**M2M D1-D2**) that implements the *Generation of KDM models* task and the transformations between D2 and D3 (**M2M D2-D3**) which implements the *Generation of CMS model task*. In the following subsections we explain the implementation of these two types of transformations.

In this subsection we present ***the implementation of the T2M transformations*** allowing the *Knowledge extraction task* and ***the implementation of the M2M transformations*** allowing the *Generation of KDM models task* and the *Generation of the CMS model task*.

3.1 Implementation of the T2M Transformations

In this subsection, we explain the activities for the implementation of the T2M transformations to extract ASTM models from PHP code. In Fig 2 they appear as *T2M D0-D1*. We use a real example of PHP code from a CMS-based Web application implemented in Drupal to illustrate the different activities performed in this implementation and to show the feasibility of our approach.

Definition of the PHP Grammar. The first activity is to define the PHP grammar using EBNF language [21]. It was one of the most tedious and time-consuming activities because we had to resolve the left-recursivity conflicts among the elements defined in the grammar. These elements are classified into two groups: 1) expressions (such as *logicalOr* or *function call*) and 2) statements (such as *assignment* or *conditional*).

Expressions are the cornerstone of the PHP language since they represent those elements which evaluate to a certain value, e.g. arithmetic expressions such as *$var+5+3* which evaluates to a number or logical expressions such as *$var or $tar* which evaluates to a logical value. Fig 3 shows the correspondence of a function definition implemented in PHP with its specification in the defined EBNF-based PHP grammar.

From the PHP grammar and using the Xtext framework we obtain automatically three artifacts: 1) a metamodel, 2) a textual editor and 3) a parser implemented in Java that allow us to recognize the elements of the PHP grammar from code written in PHP. We decided to use Xtext framework because this parser facilitates us the

implementation of the model extractor (third activity) since we use the methods implemented in this parser to recognize the PHP elements that will be mapped to the ASTM model.

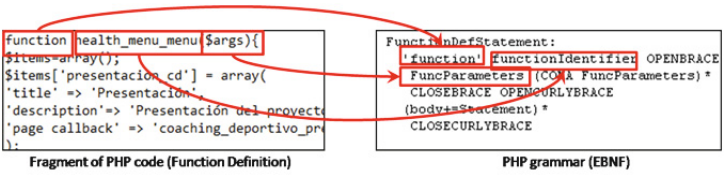


Fig. 3. Correspondence of PHP code with EBNF-based PHP grammar

Mapping PHP Grammar Elements to Elements of ASTM. In the second activity we define the mappings between the elements of the PHP grammar and the elements of ASTM. The definition of these mappings are necessary to implement the model extractor in the third activity. Some of these mappings are presented in Table 1, e.g. a *VariableDefStatement* of the PHP grammar ($var=3;) is mapped to a *VariableDefinition* of the ASTM or a *Addition* expression in the PHP grammar (9+3) is mapped to a *BinaryExpression*.

At the time of defining these mappings we realized that some elements from the PHP grammar cannot be mapped to elements of ASTM. For that reason, we extended the ASTM with the specific elements of the PHP code. Some of these elements are: *xor* operator (xor)**,** not *identical* operator (!==), *supressWarning* operator *(@)* or *instance of* operator (instanceof).

Table 1. Mapping PHP grammar elements to ASTM elements

Group	PHP Grammar element	ASTM element
Statements	VariableDefStatement	VariableDefinition
	FuncDefStatement	FunctionDefinition
Expressions	Addition	BinaryExpression
	FunctionCall	FunctionCallExpression

Otherwise, we needed to redefine existing elements of ASTM to make the mapping possible. For example, we had to redefine the attribute *condition* of the *ForStatement* element. This attribute is defined as a required and specifies the condition of a *for* statement. We had to redefine it as optional to allow map the *for* statements without condition permitted in PHP to the *ForStatement* element in ASTM.

Other redefined elements are: *swithStatement*, *compilationUnit* and *arrayAccess*. Due to space limitations we do not explain each of them.

Implementation of a Model Extractor. Finally, in the third activity we implement a parser, that we call model extractor, to obtain ASTM models from PHP code. This model extractor is implemented in Java. For its implementation we use: 1) on the one hand, the parser obtained by Xtext in the first activity to recognize the syntax

elements from code written in PHP, 2) on the other hand, the API in Java obtained automatically from ASTM by using the Eclipse Modeling Framework (EMF) [22], to generate the elements of the ASTM models.

Fig 4 shows how our model extractor identifies a function definition written in PHP and extracts a *FunctionDefinition* element in the ASTM model. As we can see in Fig 4 the fragment of PHP code conforms to the PHP grammar defined in EBNF and the model generated conform to ASTM.

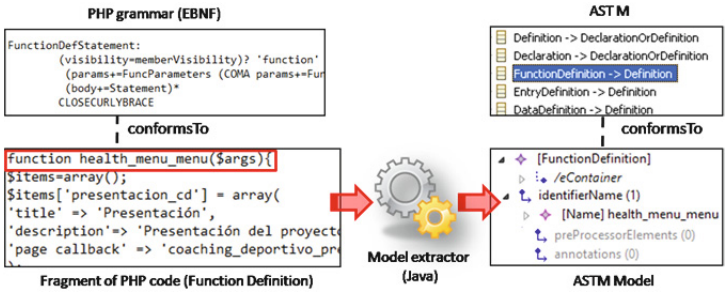


Fig. 4. Extracting a function definition in PHP to ASTM model

3.2 Implementation of the M2M Transformations

In this subsection we explain the implementation of the M2M transformations. In Fig 2, they appear as *M2M D1-D2* and *M2M D2-D3*. Firstly, we explain the transformation *M2M D1-D2* which corresponds to the implementation of the *Generation of KDM models task* and then we present the transformation *M2M D2-D3* which is the implementation of the *Generation of the CMS model task*.

Implementation of M2M D1-D2. Generation of KDM models task. In this subsection we present the definition of the M2M transformations used for the implementation of the *Generation of KDM models task*. The main goal of this task is to transfer automatically the information gathered in the ASTM models at PSM level into KDM models at PIM level. KDM models allow to model all the artifacts of the legacy system in an integrated and technological-independent manner [23].

Specifically, we use two packages of KDM, Code and Action packages, to define the KDM models which are called Code Models. Only we use these packages because the legacy source code is the unique artifact considered within our reverse engineering phase.

The Code package represents the named items from the source code and several structural relationships between them and the Action package focuses on the behavior descriptions and control- and data-flow relationships determined by them [19]. According to the Code package, the items from the source code are represented by the *AbstractCodeElement* metaclass which is specified in different types of elements such as *MethodUnits* or *ActionElements* [19]. The *ActionElement* class describes a basic unit of behavior. It has a string attribute called *kind* which specifies different kinds of actions such as, *assign*, *call method*, *add* and so on.

Table 2. Mapping ASTM elements to KDM elements

PHP element	ASTM element	KDM element
VariableDefStatement	VariableDefinition	ActionElement (kind = ArraySelect)
		ActionElement (kind = NewArray)
		ActionElement (kind = MemberSelect)
FuncDefStatement	FunctionDefinition	MethodUnit
Addition	BinaryExpression	ActionElement (kind = Add)
FunctionCall	FunctionCallExpression	ActionElement (kind = MethodCall)

Most of the mappings defined in these transformations, relate ASTM elements to *ActionElements* in the KDM model. The attribute *kind* of each *ActionElement* specifies the kind of action as we can see in Table 2.

As we can see in Table 2, the ASTM element *VariableDefinition* is mapped to three different ActionElements. Depending on the initial value of the Vari*ableDefinition* the *kind* attribute takes different values. For instance, $var=array() is a *VariableDefinition* whose initial value defines a new array so that the *kind* attribute takes the value of *NewArray* (second example of KDM elements in Table 2).

After defining the mappings of these M2M transformations we implement the transformation rules in ATL. Fig 5 shows how these transformations rules generate a *MethodUnit* in KDM model from the *FunctionDefinition* in the ASTM model.

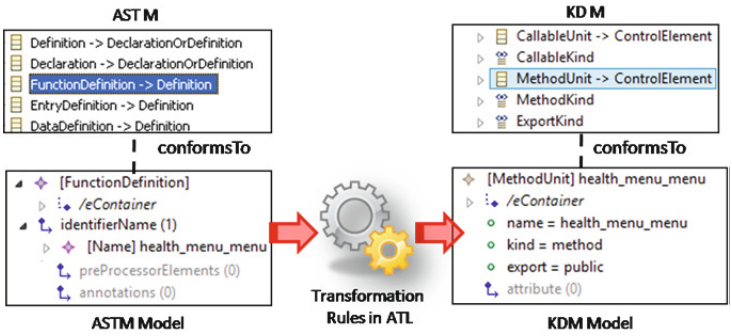


Fig. 5. Transforming a FunctionDefinition (ASTM) into a MethodUnit (KDM)

Implementation of M2M D2-D3. Generation of CMS models task. In the following, we present the implementation of the last M2M transformations (M2T D2-D3) which corresponds to the *Generation of CMS model task*. The aim of this task is to represent the information captured within the KDM models into the CMS domain using the CMS model which conforms to the CMS Common Metamodel.

These M2M transformations allow to define which pieces of the source code represented in KDM are transformed into specific elements of the CMS Model such as *modules*, *blocks* or *pages*.

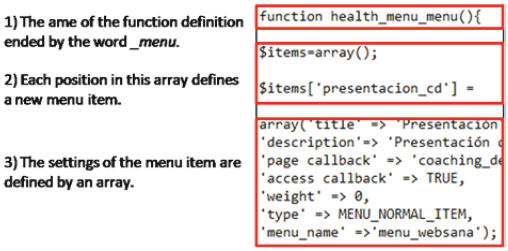


Fig. 6. Coding pattern to define menu items in Drupal

Regarding the source code and the CMS platform (e.g. Drupal, Joomla!) it is possible to establish coding patterns which allow to contextualize a piece of code within the CMS domain. For instance, in Drupal the definition of menu items (the links composing a menu) are defined by means of a function definition whose name ends with *_menu*. Within this function definition each menu item is defined as a new position in an array called $items. Moreover, the settings of a menu item are specified by the definition of a second array. Fig 6 shows the coding pattern that Drupal uses to define menu items.

Fig 7 shows how these transformations rules generate a *MenuItem* in the CMS model from the *MethodUnit* in the KDM model.

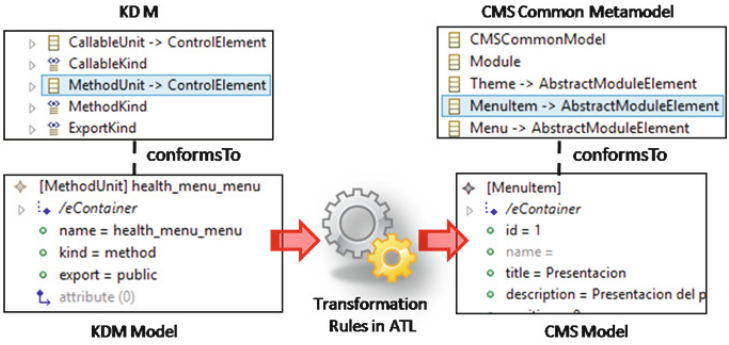


Fig. 7. Transforming a MethodUnit (KDM) into a MenuItem (CMS)

4 Related Works

In this section we present some of the ADM-based approaches found in the literature and we compare them with our ADM-based method. To analyze thoroughly these approaches we have defined seven criteria:

- **Criterion 1** is related to the three reengineering phases covered by each approach: 1) reverse engineering, 2) restructuring and 3) forward engineering.
- **Criterion 2** is related to the scope of the approach: 1) databases, 2) Web applications, 3) Web services or 4) legacy systems (general purpose).
- **Criterion 3** indicates the MDA abstraction levels considered by the approach.

- **Criterion 4** is related to the metamodels considered within the approach and the models conform to.
- **Criterion 5** is related to the technique used for the implementation of T2M transformations. Three values to establish: 1) a parser, 2) a existing tool in the market, 3) the implementation using languages, such as Gra2MoL [24].
- **Criterion 6** is related to the source code implementing the legacy system from which de approach extracts the models.
- **Criterion 7** serves to analyze the types of M2M transformations considered by the approach: vertical or horizontal [25].

Van Hoorn et al., present in [26] *DynaMod,* a method which addresses model-driven modernization of software systems which considers the three reengineering phases: reverse engineering, restructuring and forward engineering phases. The scope of this approach is not focused on a concrete context, i.e. it is a general scope. For the reverse engineering phase, it proposes to extract from Java code architectural models which conform to the KDM metamodel at PIM level. To extract these models they propose the implementation of a parser. As for the restructuring phase, it proposes to refine manually the extracted architectural models with extra information provided by system experts. Finally, as for the forward engineering phase, it defines M2M transformations to generate target architecture models for the target system. These target models also conform to the KDM metamodel (PIM level) so that we consider these M2M transformations as horizontal since the source model and the target model are defined at the same MDA level.

Sadovykh et al., present in [27] a method for migrating legacy systems implemented in C++ to target systems implemented in Java. It addresses the three reengineering phases. It is not focused in a concrete scope. As for the reverse engineering phase, it proposes to extract architectural models from the C++ code at PSM level conforming to UML. It does not specify the way to implement these T2M transformations. From this PSM model, it generates automatically a PIM model by means of M2M transformations. This PIM model also conforms to UML metamodel. Regarding the restructuring phase, it refactors the generated PIM model eliminating platform dependencies and extracting business logic. For the forward engineering phase, it generates a Java PSM model (conforming to the UML metamodel) from the PIM model by means of M2M transformations. The M2M transformations defined are considered as vertical transformations since the source models and the target models belong to a different MDA level.

Pérez-Castillo et al., present in [28] a reengineering process called *Preciso* to recover and implement Web Services in automatic manner from relational databases. This approach also addresses the three reengineering phases. Its scope is focused on the database context. Considering the reverse engineering phase, it proposes to extract models from the relational database of a legacy system. These models, at PSM level, conform to a SQL-92 metamodel. To extract these models they implement a parser. Then, the PSM model is transformed into a PIM model by means of M2M transformations. This PIM model conforms to the UML2 metamodel and it is considered the basis to define Web Services. As for the restructuring phase, the

developer can adapt the generated PIM model by refining the features of Web services. For the forward engineering phase, a PSM model conforming to the WSDL metamodel is generated from the PIM model by means of M2M transformations. The M2M transformations defined are considered as vertical transformations since the source model and the target model belong to a different MDA level.

Bruneliere et al., present in [20] *MoDisco*, an extensible approach for model-driven reverse engineering which allows extracting platform models from Java, XML and JSP code. This approach is not constraint in a specific scope, thus it can be applied to any legacy system. This approach just considers the reverse engineering phase and concretely the task of knowledge extraction. Therefore, Modisco can extract from legacy systems models at PIM level conforming to KDM as well as at PSM level conforming to a set of platform metamodels such as Java, XML and JSP metamodels. For the extraction of these models, Modisco implements a set of parsers so-called *discoverers*. This approach does not consider M2M transformations.

Pérez-Castillo et al., proposes another approach in [29] called *Marble* for recovering business processes from legacy systems. This approach addresses two of the reengineering phases: reverse engineering and restructuring phases. Its scope is not specific for a concrete context. During the reverse engineering phase, it extracts from Java code a model conforming to Java metamodel at PSM level by means of a parser. Then from this Java model, a KDM model at PIM level is generated by automatic M2M transformations. Finally, a business process model conforming to the BPMN metamodel at CIM level is obtained from the KDM model also by means of M2M transformations. In the restructuring phase, the experts can refine manually the business process model. All the M2M transformations are considered as vertical, since they allow obtaining models from different MDA levels.

Finally, **Vasilecas et al.,** presents in [30] a process which derives business rules from a legacy system. This approach addresses two of the reengineering phases: the reverse engineering and restructuring phases. The scope of this approach is general. During the reverse engineering phase, it extracts ASTM models at PSM level from the source code of the legacy system by means of parser. Then a KDM model at PIM level is generated from the ASTM model by means of M2M transformations implemented in ATL. As for the restructuring phase, business rules are identified from the KDM model. The M2M transformations are considered as vertical, since they allow obtaining models from different MDA levels.

Table 3 presents the comparison of our approach with the other approaches considering the seven criteria. Based on the results shown in table and considering the *criterion 1 (***phases***)* we can say that the half of the approaches does not consider the forward engineering phase, but all of them take into account the reverse engineering phase. Hence, we can state that the main phase for a ADM-based approach is the reverse engineering phase. Our approach considers the three reengineering phases.

Regarding the *criterion 2 (***scope***)* we conclude that most of the approaches do not have a specific scope, just *Preciso* is focused on the database context. Otherwise, we did not find any approach centered in the CMS-based Web application context like our approach.

Table 3. Comparison established with the related works

Approach	Phases	Scope	MDA levels	Metamodels	T2M	Source code	M2M
Van Hoorn et al., [26] (DynaMod)	Reverse engineering Restructuring Forward engineering	Legacy systems	PIM	KDM	Parser	Java	Horizontal
Sadovykh et al., [27]	Reverse engineering Restructuring Forward engineering	Legacy systems	PSM PIM	UML	Not specified	C++	Vertical
Pérez-Castillo et al., [28] (Preciso)	Reverse engineering Restructuring Forward engineering	Data base	PSM PIM	SQL-92 UML2 WSDL	Parser	SQL-92	Vertical
Bruneliere et al., [20] (Modisco)	Reverse engineering	Legacy systems	PSM PIM	Java, XML, JSP and KDM	Parser	Java, XML and JSP	Not specified
Pérez-Castillo et al., [29] (Marble)	Reverse engineering Restructuring	Legacy systems	PSM PIM CIM	Java, KDM and BPMN	Parser	Java	Vertical
Vasilecas et al., [30]	Reverse engineering Restructuring	Legacy systems	PSM PIM	ASTM and KDM	Parser	Not specified	Vertical
Our ADM-based method	Reverse engineering Restructuring Forward engineering	CMS-based Web applications	PSM PIM	ASTM and KDM	Parser	PHP	Horizontal and vertical

According to the *criterion 3 (***MDA levels***)* we can state that most of the approaches define models at PSM and PIM levels. Just *Dynamod* specifies models only at PIM level. Models at CIM level just are defined by *Marble*. Our approach defines models at PSM and PIM levels, but not at CIM level.

As for the *criterion 4 (***metamodels***)*, we can say that four out of the six approaches consider the standard ADM metamodels, mainly the KDM metamodel. The ASTM metamodel is just considered by Vasilecas et al., [30] at the time of extracting models from the code, the rest of approaches bet for the definition of a specific metamodel such as SQL-92 metamodel by *Preciso* or a Java metamodel by *Modisco* and *Marble*. Our approach considers both, ASTM and KDM metamodels.

Considering the c*riterion* 5 *(***T2M***)*, we conclude that the most popular technique to implement the T2M transformations is a parser. In our approach we also define a parser which we call *model extractor* to obtain ASTM models from the PHP code of a legacy CMS-based Web application.

Regarding the *criterion 6 (***source code***)*, we can state that there are approaches extracting models from Java, C++ even SQL-92, but none from PHP. Our approach is the only one that extracts models from PHP code.

Finally, as for the *criterion 7 (***M2M***)*, we can say that most of the approaches define vertical M2M transformations at the time of generating PIM models from PSM models, just *Dynamod* only defines horizontal transformations among PIM models. In our approach we define vertical M2M transformation allowing us to transform PSM models to PIM models and conversely, as well as we implement horizontal transformations when we generate the CMS model from the KDM model, both at PIM level as it is shown in Fig 1.

5 Conclusions and Future Works

Currently, many organizations experiment the necessity of migrating their CMS-based Web applications to another CMS or to a new version of the same CMS which meet better their needs. In the literature, we have not found any method to standardize and automate this reengineering process. For this reason, we proposed an ADM-based method to allow the migration of CMS-based Web applications.

This method is composed of three phases: 1) reverse engineering, 2) restructuring and 3) forward engineering. In this paper we have focused in presenting the implementation of the first phase of reverse engineering.

This reverse engineering phase comprises three tasks: 1) Knowledge extraction, in this task ASTM models are extracted from the PHP code by text-to-model (T2M) transformations implemented by a model extractor, 2) Generation of KDM models, KDM models are generated from the previous ASTM models by means of M2M transformations and 3) Generation of CMS model, from the KDM models and by M2M transformations we generate the CMS Model which conforms to the CMS Common Metamodel.

As for the implementation of the *knowledge extraction task*, we can say that the definition of the PHP grammar has been considered the most tedious and time-consuming activity because of the necessity of resolving the left-recursivity conflicts among the elements of the PHP grammar. For the implementation of the model extractor we used the parser of the PHP grammar obtained by Xtext and the API in Java of ASTM generated by its implementation in EMF.

As for the implementation of the *Generation of KDM models task* and the *Generation of CMS model task* we can say that we have implemented transformation rules using ATL.

According to the related works we can conclude that there is no any ADM approach to extract models from PHP code as well as there are a few approaches for the generation of ASTM models. We think that the low use of ASTM is because the last and unique version (1.0) was submitted rather recently, in January 2011.

Acknowledgements. This research has been partially funded by the Project MASAI (TIN-2011-22617) from the Spanish Ministry of Science and Innovation.

References

1. Boiko, B.: Understanding Content Management. Bulletin of the American Society for Information Science and Technology 28, 8–13 (2001)
2. Chikofsky, E.J., Cross, J.H.: Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software 7, 13–17
3. Abstract Syntax Tree Metamodel specification of the OMG, http://www.omg.org/spec/ASTM/1.0
4. Pérez-Castillo, R., De Guzmán, I.G.-R., Piattini, M.: Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. Computer Standards & Interfaces 33, 519–532 (2011)
5. Structured Metrics Metamodel, http://www.omg.org/spec/SMM/
6. Fischer, G., Lusiardi, J., Wolff von Gudenberg, J.: Abstract Syntax Trees - and their Role in Model Driven Software Development. In: International Conference on Software Engineering Advances (ICSEA 2007), p. 38. IEEE Computer Society, Cap Esterel (2007)
7. Trias, F., De Castro, V., López-sanz, M., Marcos, E.: A Systematic Literature Review on CMS-based Web Applications. In: ICSOFT (2013)
8. Trias, F., De Castro, V., López-Sanz, M., Marcos, E.: An ADM-based Method for migrating CMS-based Web applications. In: 25th International Conference on Software Engineering and Knowledge Engineering (SEKE), Boston, EUA (2013)
9. Drupal CMS, http://drupal.org/
10. Joomla! CMS, http://www.joomla.org/
11. Wordpress CMS, http://wordpress.org/
12. Ric Shreves: Open Source CMS Market Share (2008)
13. Xtext project, http://www.eclipse.org/Xtext
14. Trias, F.: Building CMS-based Web Applications Using a Model-driven Approach. In: Sixth International Conference on Research Challenges in Information Science (RCIS), pp. 1–6 (2012)
15. Atlas Transformation Language, http://www.eclipse.org/atl/
16. Fleurey, F., Breton, E., Baudry, B., Nicolas, A.: Model-Driven Engineering for Software Migration in a Large Industrial Context. Engineering, 482–497 (2007)
17. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: Model-Driven Architecture. In: Bruel, J.-M., Bellahséne, Z. (eds.) Advances in Object-Oriented Information Systems, pp. 233–239. Springer, Heidelberg (2002)
18. Vidgen, R., Goodwin, S., Barnes, S.: Web Content Management. In: 14th Bled Electronic Commerce Conference, Slovenia (2001)
19. Pérez-Castillo, R., De Guzmán, I.G.-R., Piattini, M.: Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. Computer Standards & Interfaces 33, 519–532 (2011)
20. Barbier, G., Bruneliere, H., Jouault, F., Lennon, Y., Madiot, F.: MoDisco, a model-driven platform to support real legacy modernization use cases. Information Systems Transformation: Architecture-Driven Modernization Case Studies, 365 (2010)
21. ISO/IEC 14977:1996 - EBNF, http://www.iso.org/iso/catalogue_detail.htm?csnumber=26153
22. Bezivin, J., Jouault, F., Brunette, C., Chevrel, R., Kurtev, I.: Bridging the Generic Modeling Environment (GME) and the Eclipse Modeling Framework (EMF). In: International Workshop on Best Practices for Model Driven Software Development, San Diego, California, USA (2005)

23. Pérez-Castillo, R., De Guzmán, I.G.-R., Piattini, M., Places, Á.S.: A case Study on Business process recovery using an e-government sytem. Software - Practice and Experience 42, 159–189 (2011)
24. Cánovas Izquierdo, J.L., Sánchez Cuadrado, J., García Molina, J.: Gra2MoL: A domain specific transformation language for bridging grammarware to modelware in software modernization. In: MODSE, pp. 1–8 (2008)
25. Mens, T., Gorp, P., Van: A taxonomy of model transformation. Electronic Notes Theor. Comput. Sci. 152, 125–142 (2006)
26. Van Hoorn, A., Sören, F., Goerigk, W., Hasselbring, W., Knoche, H., Köster, S., Krause, H., Porembski, M., Stahl, T., Steinkamp, M., Wittmüss, N.: DynaMod Project: Dynamic Analysis for Model-Driven Software Modernization. Engineering, pp. 1–2
27. Sadovykh, A., Vigier, L., Hoffmann, A., Grossmann, J., Ritter, T., Gomez, E., Estekhin, O.: Architecture Driven Modernization in Practice: Study Results. In: 2009 14th IEEE International Conference on Engineering of Complex Computer Systems, pp. 50–57 (2009)
28. Castillo, R.P., García-rodríguez, I.: PRECISO: A Reengineering Process and a Tool for Database Modernisation through Web Services. In: Proceedings of the 2009 ACM Symposium on Applied Computing, pp. 2126–2133. ACM, New York (2009)
29. Pérez-castillo, R., Fernández-ropero, M., Guzmán, I.G., De, P.M.: MARBLE: A Business Process Archeology Tool. In: 27th IEEE International Conference on Software Maintenance, pp. 578–581 (2011)
30. Vasilecas, O., Normantas, K.: Deriving Business Rules from the Models of Existing Information Systems, pp. 95–100 (2011)

Author Index