IFIP AICT 351

Jolita Ralyté
Isabelle Mirbel
Rébecca Deneckère
(Eds.)

Engineering Methods in the Service-Oriented Context

4th IFIP WG 8.1 Working Conference
on Method Engineering, ME 2011
Paris, France, April 2011
Proceedings

Springer

IFIP Advances in Information and Communication Technology 351

IFIP – The International Federation for Information Processing

IFIP was founded in 1960 under the auspices of UNESCO, following the First World Computer Congress held in Paris the previous year. An umbrella organization for societies working in information processing, IFIP's aim is two-fold: to support information processing within its member countries and to encourage technology transfer to developing nations. As its mission statement clearly states,

> *IFIP's mission is to be the leading, truly international, apolitical organization which encourages and assists in the development, exploitation and application of information technology for the benefit of all people.*

IFIP is a non-profitmaking organization, run almost solely by 2500 volunteers. It operates through a number of technical committees, which organize events and publications. IFIP's events range from an international congress to local seminars, but the most important are:

- The IFIP World Computer Congress, held every second year;
- Open conferences;
- Working conferences.

The flagship event is the IFIP World Computer Congress, at which both invited and contributed papers are presented. Contributed papers are rigorously refereed and the rejection rate is high.

As with the Congress, participation in the open conferences is open to all and papers may be invited or submitted. Again, submitted papers are stringently refereed.

The working conferences are structured differently. They are usually run by a working group and attendance is small and by invitation only. Their purpose is to create an atmosphere conducive to innovation and development. Refereeing is less rigorous and papers are subjected to extensive group discussion.

Publications arising from IFIP events vary. The papers presented at the IFIP World Computer Congress and at open conferences are published as conference proceedings, while the results of the working conferences are often published as collections of selected and edited papers.

Any national society whose primary activity is in information may apply to become a full member of IFIP, although full membership is restricted to one society per country. Full members are entitled to vote at the annual General Assembly, National societies preferring a less committed involvement may apply for associate or corresponding membership. Associate members enjoy the same benefits as full members, but without voting rights. Corresponding members are not represented in IFIP bodies. Affiliated membership is open to non-national societies, and individual and honorary membership schemes are also offered.

Jolita Ralyté Isabelle Mirbel
Rébecca Deneckère (Eds.)

Engineering Methods in the Service-Oriented Context

4th IFIP WG 8.1 Working Conference
on Method Engineering, ME 2011
Paris, France, April 20-22, 2011
Proceedings

Springer

Volume Editors

Jolita Ralyté
Université de Genève, Centre Universitaire d'Informatique
Battelle, bâtiment A, 7, route de Drize, 1227 Carouge, Switzerland
E-mail: jolita.ralyte@unige.ch

Isabelle Mirbel
Université Nice-Sophia Antipolis, Département Informatique
Parc Valrose, 06108 Nice Cedex 02, France
E-mail: isabelle.mirbel@unice.fr

Rébecca Deneckère
Université Paris 1, Centre de Recherche en Informatique
90 rue de Tolbiac, 75013 Paris, France
E-mail: rebecca.deneckere@univ-paris1.fr

ISSN 1868-4238 e-ISSN 1868-422X
ISBN 978-3-642-19996-7 e-ISBN 978-3-642-19997-4
DOI 10.1007/978-3-642-19997-4
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011923072

CR Subject Classification (1998): D.2, H.4, H.5, 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

Over the last two decades the discipline of method engineering has evolved from simple ad-hoc method construction to situational and domain-specific method engineering approaches as a response to the increasing complexity and diversity of software and information systems developments. Several theories, approaches and tools have been proposed to support the construction of project-specific information system development methods where each method would be based on the particular project situation and requirements. To attain such a high degree of flexibility, methods are understood to be modular, built from so-called method fragments or method chunks, which are stored in method repositories and can be assembled in situation-specific methods.

Despite the great advance in this domain, many issues are still open for fundamental research. The notion of situation, its characterization and evaluation as well as the suitability of method fragments to the situation have been investigated but still need more theory and experimentation. How to evaluate the quality of a newly constructed method? What is the best granularity of method fragments and method chunks? How to guide assembly of method fragments? All these questions still need an answer.

Furthermore, the evolution of enterprise software and information systems and especially their shift toward service-oriented architectures demands new ways of working, thinking and designing systems that we now call service-oriented systems. New methods, techniques and tools based on the concept of service and better fitting the current development situations are under development and experimentation and are the main topic of this volume. Besides, the notion of service is also emerging in the domain of method engineering as a new type of method building block and therefore becomes a new fundamental concept of the discipline.

Engineering methods, techniques and tools for the analysis, design and evolution of information systems is one of the main research areas of the IFIP Work Group 8.1. Successful Working Conferences have been organized on this topic in Atlanta in 1996, in Kanazawa in 2002 and in Geneva in 2007. A new edition of the IFIP WG 8.1 Working Conference on Method Engineering with a subtitle "Engineering Methods in the Service-Oriented Context" was held at the University of Paris 1 – Pantheon Sorbonne, in France, during April 20–22, 2011.

The 19 papers (13 full papers and 6 short papers) included in this volume were carefully selected by an international Program Committee out of 30 submissions. Each submission was evaluated by three Program Committee members, recruited from IFIP WG 8.1 members and other researchers active in the method engineering field. The overall quality of the papers was high and very well fitting to the scope of the conference.

The conference program featured two keynote talks by renowned method engineering researchers: Naveen Prakash from MRCE, Faridabad, India, presented "An Assessment of Method Engineering," while Marko Bajec from the University of Ljubljana, Slovenia, discussed the "Application of Method Engineering Principles in Practice." Moreover, a tutorial on "Creating Self-Describing Method Component Repositories with ISO/IEC 24744" was given by Cesar Gonzalez-Perez from the Spanish National Research Council. The format of a working conference provided the participants with an opportunity to have extensive and interactive paper discussions in plenary sessions.

We wish to thank the members of the international Program Committee and the additional reviewers for their valuable and professional work in crafting a high-quality program for this conference. A special word of thanks goes to the keynote speakers and the tutorial lecturer for their willingness to present the latest views and achievements in the discipline. We finally would like to thank all the participants and the conference organizers for their valuable contributions.

We wish you a pleasant reading and a fruitful use of these research results in your research and applications.

April 2011

Jolita Ralyté
Isabelle Mirbel
Rébecca Deneckère

Conference Organization

General Conference Chair

Jolita Ralyté	University of Geneva, Switzerland

Program Committee Chair

Isabelle Mirbel	University of Nice Sophia Antipolis, France

Organizing Chair

Rébecca Deneckère	University of Paris 1 – Panthéon Sorbonne, France

Program Committee

Pär Ågerfalk	Sweden
David Avison	France
Marko Bajec	Slovenia
Sjaak Brinkkemper	The Netherlands
Albertas Čaplinskas	Lithuania
Corine Cauvet	France
Massimo Cossentino	Italy
Xavier Franch	Spain
Cesar Gonzalez-Perez	Spain
John Grundy	Australia
Remigijus Gustas	Sweden
Frank Harmsen	The Netherlands
Peter Haumer	USA
Brian Henderson-Sellers	Australia
Charlotte Hug	France
Manfred Jeusfeld	The Netherlands
Paul Johannesson	Sweden
Fredrik Karlsson	Sweden
Steven Kelly	Finland
John Krogstie	Norway
Susanne Leist	Germany
Michel Léonard	Switzerland
Mauri Leppanen	Finland
Pericles Loucopoulos	UK
Kalle Lyytinen	USA

Haralambos Mouratidis	UK
Leon J. Osterweil	USA
Oscar Pastor	Spain
Juan Pavón	Spain
Anne Persson	Sweden
Yves Pigneur	Switzerland
Naveen Prakash	India
Erik Proper	The Netherlands
Iris Reinhartz Berger	Israel
Dominique Rieu	France
Colette Rolland	France
Motoshi Saeki	Japan
Guttorm Sindre	Norway
Keng Siau	USA
Juha-Pekka Tolvanen	Finland
Inge van de Weerd	The Netherlands
Robert Winter	Switzerland
Boštjan Žvanut	Slovenia

Additional Referees

Sophie Dupuy-Chessa	France
Boris Fritscher	Switzerland
Agnès Front	France
Kevin Vlaanderen	The Netherlands

Table of Contents

Keynote Talks

Situated Method Engineering

Method Engineering Foundations

Customized Methods

Tools for Method Engineering

New Trends to Build Methods

Method Engineering for Services

An Assessment of Method Engineering

Naveen Prakash

MRCE, Sector 43, Aravali Hills, Badhkal Surajkund Road
Faridabad 121001, India
praknav@hotmail.com

The area of method engineering has been researched extensively in the last two decades. The first exclusive conference in the subject was held in 1996. In this conference a number of major strands of work and possible directions for the future were discussed. Indeed, work in almost all these directions has progressed in the last fifteen years. There is now some need to assess the work done and chart out future courses of action. Accordingly, this talk is organized in two parts, where we are and where we can go.

In the first part, starting from the initial motivations of method engineering, we shall take stock of what was promised and what has been achieved. Indeed, method engineering has introduced a number of key notions: the product and process aspects of methods, meta modeling, CAME, method rationale, situational method engineering etc. We shall bring out the progress made in developing these notions.

In the second part of this talk, we shall express our view that the future belongs to flexible and adaptable method engineering. We take an analogy with adaptability and configurability in software engineering and outline a framework for engineering adapted methods.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, p. 1, 2011.

Application of Method Engineering Principles in Practice: Lessons Learned and Prospects for the Future

Marko Bajec

University of Ljubljana, Faculty of Computer and Information Science
Head of the Laboratory for Data Technologies
Trzaska 25, 1000 Ljubljana, Slovenia
marko.bajec@fri.uni-lj.si

It seems that in IT sector we are all aware that for the development of non-trivial software the use of software methods is very important. They provides as with knowledge and guidance for the development process which otherwise might become too chaotic and out of control. It has been empirically proven that software development companies which have successfully established their software processes are more efficient, produce software of higher quality and have shorter time-to-market period; specifically if they are able to adapt their ways of working to specifics of a particular project.

In the research community Method Engineering (ME) principles have been promoted as a way to make software development methods agile and adaptable to particular circumstances, i.e. specifics of a development team and project. Unfortunately, however, ME have never been really accepted or widely used in practice. The reasons are several, not all are equally important.

At the University of Ljubljana we have done our own research to see what we can do to motivate software companies in employing ME principles. The research project was carried out under the umbrella of the Centre of excellence for "Information and Communication Technologies" with a mission to improve software development practice in Slovenian companies. The project was co-founded by the Slovenian Ministry of Higher Education, Science and Technology, European Commission and five participating Software Companies.

To reach the goal our idea was to facilitate the companies with a framework and tool-support for reengineering their ways of working, so that the gap between their official methods (documented methods they claim to follow) and the ways how they actually develop software would be as small as possible. As a part of this framework we have developed our own approach for process configuration (PCA) that suggests how to incorporate flexibility into formalised or documented methods, so that they could be adjusted to suite best to circumstances of a particular project. The PCA tells how to describe the ways of working in an organisation (organisation's base method) so that project-specific methods could be than created automatically by using appropriate tool-support.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 2–3, 2011.

It has been now three years after the participating companies incorporated the framework and supporting tools into their environments. In this talk I would like to provide the audience with more information on the research project that we have performed and share the lessons we have learned. Our findings lead us to not very enthusiastic conclusions and force us to look for different ways to tackle the problem.

Incremental Method Engineering for Process Improvement – A Case Study

Dominique Mirandolle, Inge van de Weerd, and Sjaak Brinkkemper

Utrecht University, Department of Computer Science,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
d.e.mirandolle@students.uu.nl,
{i.vandeweerd,s.brinkkemper}@cs.uu.nl

Abstract. In order for companies to improve the maturity level of their development process, they need to design new methods or adapt the existing ones. This research aims to deliver a proof of concept of how incremental method engineering supports the maturation of methods in a product software company. We show how the adaptation of a method can lead to a higher maturity level. We assessed the method of a case company by means of the situational assessment method, resulting into the maturity level and situational factors. We also modeled eight different prioritization methods according to their maturity level and situational factors to find out which of these could be implemented into the case company's method in order to evolve to a higher maturity level. After matching the situational factors of the available methods with the company factors we find one method that is suitable to implement into the existing method at the case company. We explain how the implementation can take place and how this would evolve the method to full maturity.

Keywords: Incremental method engineering, software product management, competence model, situational factors, maturity matrix.

1 Introduction

Product software companies have to be on track with the latest changes in the field of software development and product management. Naturally, their processes and methods need to be adjusted accordingly to the changes in the environment and growth of the company. Yet, many product software companies find it difficult to improve the maturity level of their methods [1]. In order for companies to improve this maturity level they need to design new methods or adapt the existing ones, while there is little education available in the software product management (SPM) area [2].

In this research, we use an incremental method engineering approach to improve an organization's process maturity. Method engineering (ME) is the discipline to design, construct and adapt methods, techniques and tools for the development of information systems [3]. If a method is tuned to the project at hand, this is called situational ME. When only a method fragment, and not the entire method, is changed, this is called incremental ME. A method increment can be defined as “a method

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 4–18, 2011.

adaptation, in order to improve the overall performance of a method" [1]. Incremental ME can be seen as a sub type of situational ME, where incremental ME focuses more on evolving a method in time towards a higher maturity level by changing small parts of the method.

1.1 Aim of This Research

The aim of this study is to deliver a proof of concept of how incremental ME supports the maturing, and thus the improvement of processes in a product software company. The main research question in this study is formulated accordingly:

"How can incremental method engineering support process improvement in the software industry?"

By answering this question we aim to contribute to the field of incremental ME by showing how the adaptation of a method can lead to a higher maturity level of that method. We elaborate on how method increments, based on Situational Factors (situational factors) of both the method and the company, can evolve the method of our case company. This verifies the theoretical description of incremental ME.

1.2 Related Work

Several approaches have been introduced to make it easier for companies to change their development methods [4, 5, 6]. To help companies select a proper approach to adapt an existing method, Ralyté et al. [7] present a generic model for situational method engineering. In their approach, the method engineer is able to combine the approaches that fit the ME project the best by setting intentions (goals) and connect these with strategies. Ågerfalk et al. [8] also present a method to help method engineers with the configuration and adaptation of methods. They propose the use of premade reusable configurations of a base method suitable for a specific characteristic of a development situation. Rossi et al. [9] claim that method users, but especially method engineers need to be aware of the rationale of the method in order to coordinate the development and evolution of an existing method base.

Van de Weerd et al. use the concept of *incremental method engineering* as a principle in their Product Software Knowledge Infrastructure (PSKI) [1]. Incremental method engineering is a specific type of situational method engineering, where development methods are over time incrementally adapted to the changing conditions. The principle is used in the PSKI, which enables organizations to acquire a custom-made advice to improve their processes incrementally. An important part of the PSKI is the method base, which is loaded with existing method fragments. Accordingly to the Situational Factors (situational factors) of the company, method fragments are chosen out of the method base in order to create a more mature method. The domain for which the PSKI is initially proposed is Software Product Management. By developing the Software Product Management Competence Model [10] (Fig. 1), they give an overview and structure to the software product management domain. The model divides the internal functions of software product management into four business functions: portfolio management, product planning, release planning and

requirements management, which contain a total of 15 focus areas, such as 'requirements prioritization' and 'product roadmapping'. Furthermore, a maturity matrix for SPM was developed, in which for each focus area three to five capabilities were defined. The maturity matrix is depicted in Table 1. If all are implemented, full maturity is reached. The methods we analyzed in this research are all requirements prioritization methods, in the business function release planning.

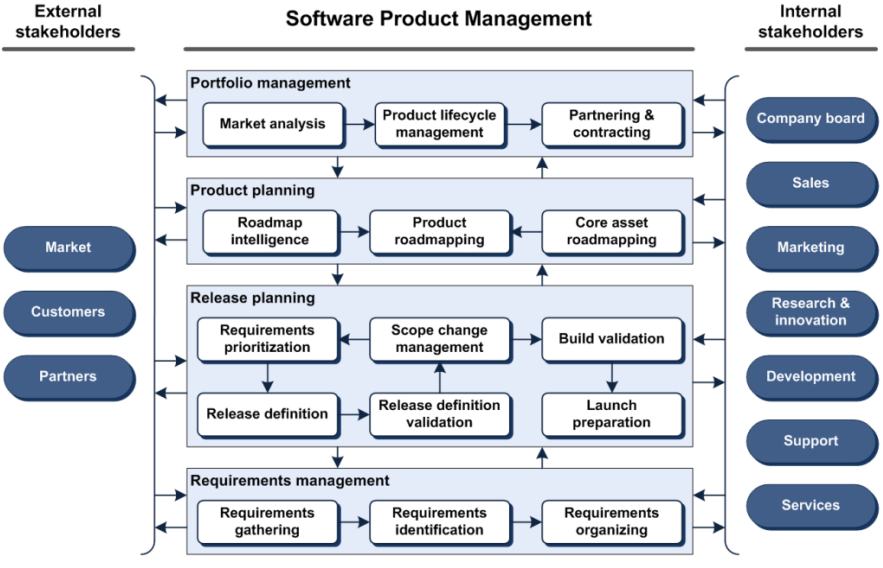


Fig. 1. SPM Competence Model

2 Research Approach

In order to deliver the proof concept and answer the main research question, we perform a case study [11] at a product software company called Teezir, hereafter called 'the case company'. We focus on a small part of the release planning stage in the SPM Competence Model: requirements prioritization. We have chosen this particular process as it has not yet reached the highest maturity level within the case company, according to the assessment. Additionally, we are familiar with several methods applicable in this stage and literature about this stage. In this stage, the requirements for an information system are sorted according to importance for certain stakeholders. The literature we use for this research mainly consists on literature about requirements prioritization methods and method engineering.

Our research approach consists of two main steps:

1. *Select and analyze methods.* In order to deliver a proof of concept we analyze eight requirements prioritization methods. The analysis is performed by measuring their maturity according to the Competence Model, developed by Bekkers & van de Weerd [10]. Additionally, we describe the situational factors per method, based on work by Bekkers et al. [12].

2. *Case study.* We analyze the prioritization method that is used by the case company, as well as its maturity levels and situational factors. Based on this information, we map the method fragments found in the previous step to the case company and select the best one. Finally, we propose how to implement this method fragment.

3 Selection and Analysis of Methods

The methods that are selected for this research are from the SPM domain, in particular focusing on requirements prioritization. Different ways exist to prioritize requirements. Some existing methods are very complex and involve many stakeholders, while others are simple. In this section, first the requirements prioritization focus area is further analyzed. Then, an overview of the selected prioritization method is presented. Finally, the situational factors of each method are listed.

3.1 Requirements Prioritization

Table 1 presents the SPM maturity matrix, consisting of the 15 focus areas, each with its own number of specific maturity levels. The focus area specific maturity levels are represented by the letters A-F in Table 1 and range from maturity level 1 to 10 (the topmost row in Table 1). In this research we focus on requirements prioritization. This focus area contains five capabilities (denoted with letters A-E) [10].

The five requirements prioritization capabilities and their goals are:

A. *Internal stakeholder involvement*
   Goal: Improved product quality & increased involvement of internal stakeholders in the product management process.
   Action: All relevant internal stakeholders indicate the requirements that should be incorporated in future releases by assigning priorities to the requirements.

B. Prioritization method
   Goal: Structure the requirement prioritization process and therewith provide a solid prioritization which is balanced and clear to all parties involved.
   Action: A structured technique is used.

C. Customer involvement
   Goal: Incorporation of customer needs and wishes in the product.
   Action: Customers and prospects indicate the requirements that should be incorporated in future releases by assigning priorities to the requirements from their point of view. Customers can also be represented by delegates.

D. Cost revenue consideration
   Goal: Create a financial basis for the prioritization.
   Action: Information about costs and revenues of each (group of) requirement(s) is taken into account during the requirements prioritization (costs can be expressed in other means than money).

E. Partner involvement
   Goal: Improved product quality & increased involvement of external stakeholders in the product management process.
   Action: Partner companies indicate requirements that should be incorporated in future releases by assigning priorities to the requirements.

Table 1. SPM Maturity Matrix

	0	1	2	3	4	5	6	7	8	9	10
Requirements management											
Requirements gathering		A		B	C		D	E	F		
Requirements identification			A			B		C			D
Requirements organizing				A		B		C			
Release planning											
Requirements prioritization			A		B	C	D			E	
Release definition			A	B	C				D		E
Release definition validation					A			B		C	
Scope change management				A		B		C		D	
Build validation					A			B		C	
Launch preparation		A		B		C	D		E		F
Product planning											
Roadmap intelligence				A		B	C		D	E	
Core asset roadmapping					A		B		C		D
Product roadmapping			A	B			C	D		E	
Portfolio management											
Market analysis					A		B	C	D		E
Partnering & contracting						A	B		C	D	E
Product lifecycle management					A	B			C	D	E

3.2 Requirements Prioritization Methods

The eight requirements prioritization methods were selected based on the availability of literature of each method. The methods are shown in Table 2, along with the capabilities that are implemented in them. When combining the information available in Table 1 and Table 2, we can see that for example the Binary Priority List method has a maturity level of 4. We can conclude this, since Table 2 shows that capability A and B are implemented in this method. According to Table 1, an implementation up to capability B has a maturity level score of 4.

Table 2. Requirements Prioritization Methods

Method	Implemented capabilities				
Binary Priority List [14]	A	B			
WinWin requirements negotiation model [15]	A	B	C		
Integer linear programming approach [16]	A	B	C	D	E
Requirements Triage [17]	A	B	C		E
MOSCOW [18]	A	B	C	D	
Cost Value Approach [19]	A	B	C	D	
Quality Function Deployment [20]	A	B	C		
Features Prioritization Matrix [21]	A	B	C	D	E

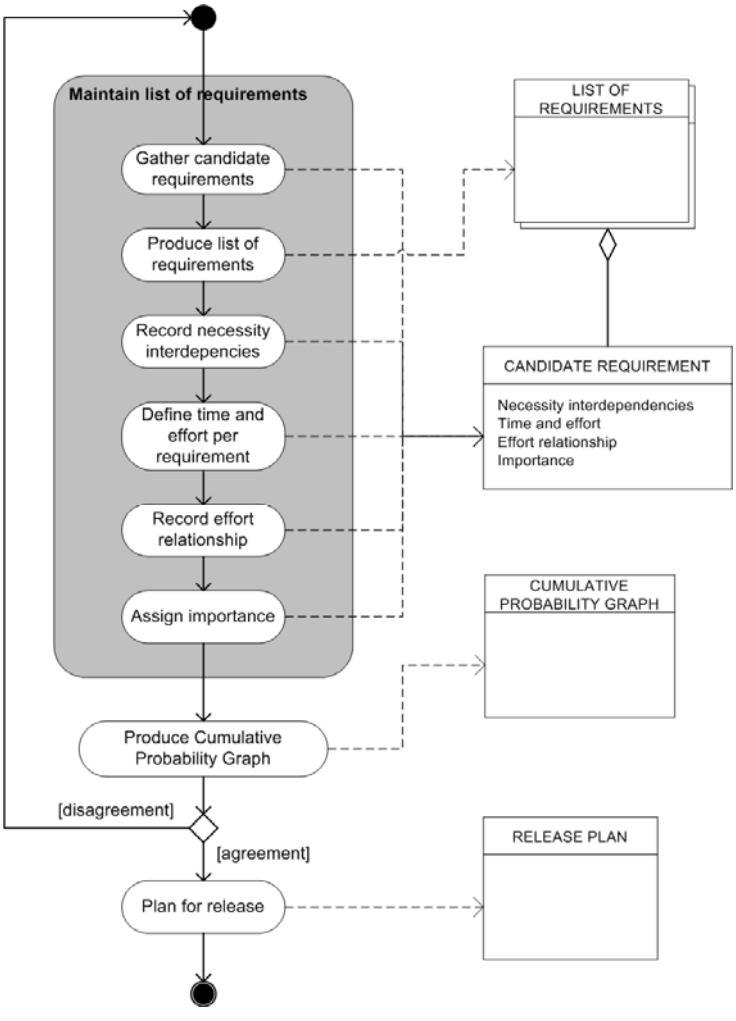


Fig. 2. PDD of the Requirements Triage method

When methods miss a capability, the maturity level only scores up to the capability before the first missing capability. For the Requirements Triage method [14] this means that we measure the maturity level only up to capability C. This results in a maturity level of 5.

We modeled all the selected methods in Process Deliverable Diagrams (PDD). A PDD is a diagram that integrates an activity diagram on the left-hand side and a deliverable view on the right-hand side [13]. To illustrate our research method, Fig. 2 shows the activities and deliverables of the Requirements Triage method [17]. For all

eight requirements prioritization methods such a PDD and a corresponding concept table and activity table were created.

3.3 Situational Factors

Bekkers [12] presents, in a case study among 14 software product companies, a list of 31 situational factors which need to be kept in mind when configuring or choosing a development method. For each of the selected methods we have marked whether the value of the situational factor is of any importance and if so, what value would suit the method best. As an example, we show the results of this analysis for Requirements Triage in Table 3. Out of the 31 situational factors, 11 are of importance in this method. This is because Requirements Triage focuses on cooperation between business and development departments. The situational factors of which the value does not affect that functionality of the method, which can contain any value, are left out in this table.

Table 3. Situational Factors of the Requirements Triage method

Situational factor	Value
Size of business unit team	Large
Size of development team	Small to Medium
Number of customers	High
Number of end-users	High
Release frequency	High
Variability of feature requests	Large
Product size	Large
Company policy	High
Customer involvement	High
Legislation	Strict
Partner involvement	High

4 Case Study

4.1 Case Study Design

The in-depth investigation in this case study takes place a product software company called Teezir, a ‘search solutions’ company (hereafter called ‘the case company’. Their main product is a web based dashboard that integrates various widgets containing representations of the online reputation of a specific brand name (for example term clouds, sentiment analysis, volume of mentions etc.). The dashboard is a standard product which can be customized by the client himself. By dragging and dropping the preferred widgets on the dashboard, a suitable application for the situation or customer at hand can be generated.

By carrying out interviews at the case company, we create a complete overview of the used prioritization method, situational factors and maturity level. This overview enables us to select the best fitting candidate method that, once implemented, will bring the case company to a higher maturity level.

Fig. 3 illustrates how the process of fitting methods works. The process of comparing the situational factors of the candidate method to the case company's method can be seen as trying to fit a key on a keyhole. In a way we have created a keyhole by defining the situational factors of the case company. The different values of the situational factors are the holes in the keyhole's cylinder. All the candidate methods are keys, of which the situational factors are pins that need to match into the holes of the keyhole's cylinder. The situational factors of the candidate methods need to be as equal as possible to those of the case company (however, not all situational factors are relevant in this case). All we need to do is find the key that matches the keyhole.

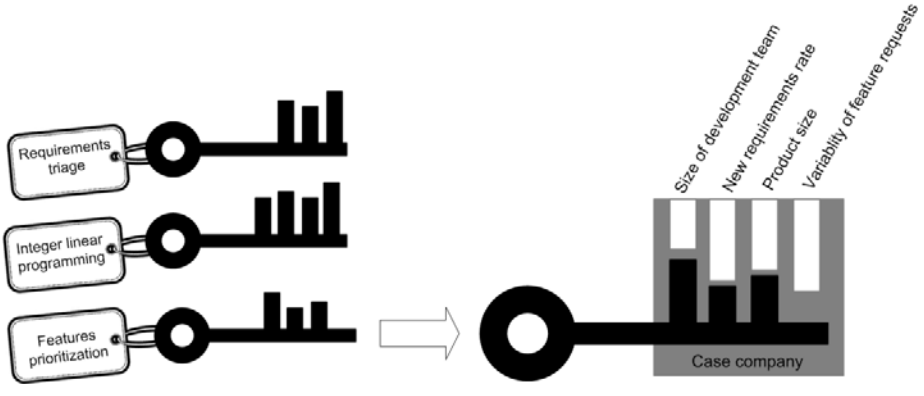


Fig. 3. Fitting the candidate methods to the case company's method

4.1.1 Conduction of Case Study

In a semi-structured interview with the case company's software engineer, we analyzed the requirements prioritization method that is used at this company. We described their method and visualized it in a Process Deliverable Diagram (PDD): a diagram that integrates an activity diagram on the left-hand side and a deliverable view on the right-hand side [13]. With this information we were able to define the maturity level of the case company nowadays. Additionally, we elaborated on the case company's situational factors. Once we knew at which maturity level the case company was operating and which situational factors influence the company, we could suggest them to adopt (one of the) method fragments we analyzed in order for them to grow and develop their method towards a higher maturity level. Finally, the method fragment which suited the case company best is implemented in the original method. We elaborated on how this can take place and visualized the matured method in a PDD.

4.1.2 Analysis of Case Study Evidence

The case company uses the Dynamic Systems Development Method (DSDM) [22]. The requirements prioritization method that is used in DSDM (and by the case company) is the MOSCOW method, as depicted in Fig. 4.

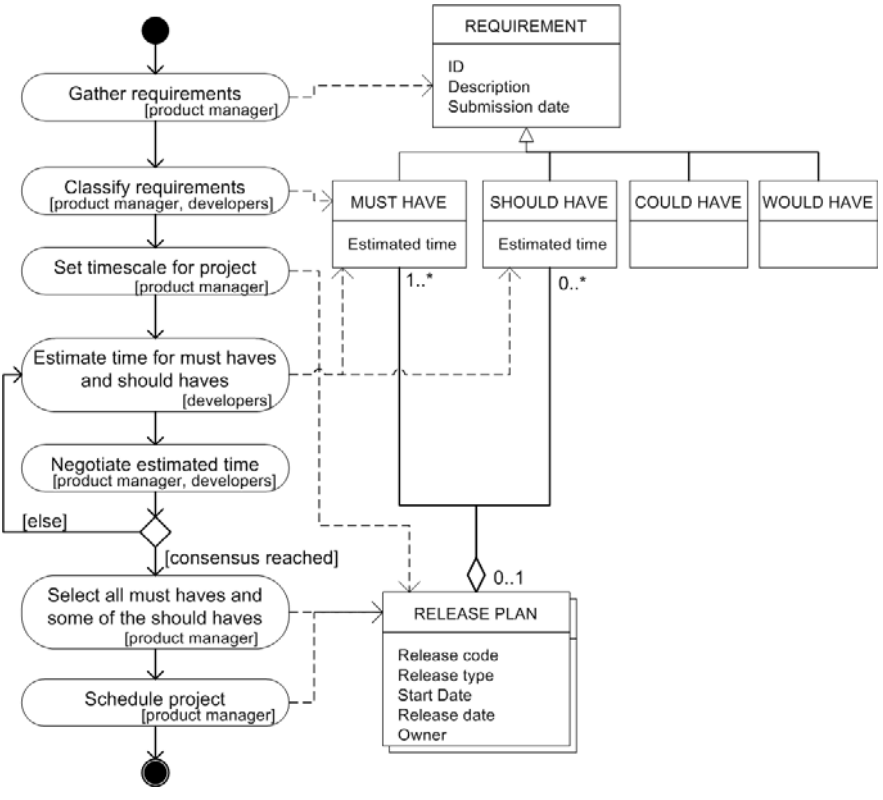


Fig. 4. PDD of the MOSCOW method

The MOSCOW requirements prioritization method contains maturity levels A to D, since it contains internal stakeholder involvement, a requirements prioritization method, customer involvement and a cost revenue consideration (measured in time).

In addition, we have analyzed the situational factors for the case company, as is presented in Table 4.

The requirement prioritization method used at the case company nowadays is MOSCOW. This method contains the maturity levels A to D. If the case company's method would evolve, activities that contain level E should be added to the method. Level E contains partner involvement, and its goal is to improve product quality and to increase involvement of external stakeholders in the product management process.

Table 4. Situational factors of the case company

Situational factor	Value
Development philosophy	Iterative
Size of business unit team	6 FTE
Size of development team	4 FTE
Customer loyalty	High
Customer satisfaction	6 (out of 10)
Customer variability	40% of customers have customized features
Number of customers	25
Number of end-users	150
Type of customers	All sorts of companies
Hosting demands	Central hosting services
Localization demand	Low
Market growth	Growing
Market size	3500+ potential customers
Release frequency	Every 250 days
Sector	Marketing
Standard dominance	Medium request for market standards
Variability of feature requests	Low
Application age	2 years
Defects per year: total	0 per year
Defects per year: serious	0 per year
Development platform maturity	Fully developed
New requirement rate	3 requests per year
Number of products	1
Product lifetime	3 year
Product size	350 KLOC
Product tolerance	High (not sensitive to bugs)
Software Platform	.NET
Company policy	High level of influence
Customer involvement	Medium involvement
Legislation	Loose
Partner involvement	High level of influence

Of the eight methods we analyzed in this research, three contain activities that implement maturity level E. These are Requirements Triage [17], Integer linear programming approach [16], and Features Prioritization Matrix [21]. Based on the situational factors of these three methods and those of the case company, we can now define which of these would suit the case company's method best (which key fits in the keyhole). Table 5 shows the values of the situational factors (the pins of the keys) of the three mature methods. We already defined the situational factors of the case company (the keyhole) in Table 4. The bottom rows of Table 5 show how many pins of the candidate methods' keys fit into the keyhole, and thus how many situational factors match to the situational factors of the case company.

The situational factors of which the value does not affect the functionality of the method and can contain any value, are left blank in this table. Additionally, we printed the situational factors that do not match the case company in italic. The cells that contain plain text do match the situational factors of the case company. At the bottom of the table we sum up how many matches and mismatches each method contains.

Table 5. Situational Factors of the three A-E methods

Situational factor	**Requirements Triage**	**Integer linear programming**	**Features Prioritization**
Development philosophy			
Size of business unit team	*Large*		
Size of development team	Small to medium		Small to medium
Customer loyalty			
Customer satisfaction			
Customer variability			
Number of customers	*High*		
Number of end-users	*High*		
Type of customers			
Hosting demands			
Localization demand			
Market growth			
Market size			
Release frequency	*High*		
Sector			
Standard dominance			
Variability of feature requests	*Large*	*Large*	
Application age			
Defects per year: total			
Defects per year: serious			
Development platform maturity		*High*	
New requirement rate	*High*	*High*	Low to medium
Number of products		*High*	
Product lifetime			
Product size	*Large*	*Large*	Small to medium
Product tolerance			
Software Platform			
Company policy	High		
Customer involvement	*High*	*High*	*High*
Legislation	*Strict*		
Partner involvement	High	High	High
Matches	22	25	30
Mismatches	9	6	1

In Table 5 it can be seen that Wiegers' Features Prioritization Matrix is not dependent on a lot of factors. The method is known to be applicable on almost every kind of project. Requirements Triage is on the other hand suitable for projects with eleven specific situational factors. Requirements Triage focuses on projects in which large amounts of requirements are involved. The method is designed specifically to deal with a 'chaos' of requirements, since it originates from the medical domain, where patients need to be 'sorted' or 'triaged' as quickly as possible. Additionally, it tries to involve as many stakeholders as possible (e.g. customers, developers, financial and legal representatives, etc.), which explains why company policy, customer involvement, legislation and partner involvement all have a high influence on the method. This suggests that the method deals with large projects, in which a large number of end-users are involved.

On the other hand, the Integer Linear Programming approach and Requirements Triage have six and nine mismatching situational factors respectively. They are both suitable for large projects, with a large amount of products involved. Therefore, it seems obvious to choose the Features Prioritization Matrix method to expand the case company's current method to maturity level E.

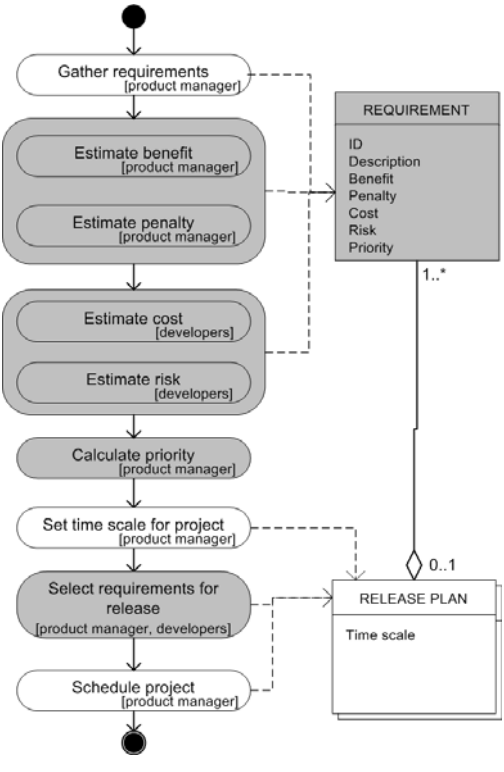


Fig. 5. Method increment with prioritization using Wiegers' matrix

The case company could evolve its requirements prioritization method by applying the multiple stakeholder sheet in the MOSCOW method. This would mean that all stakeholders, including partners, would be involved in the requirements prioritization method. If the multiple stakeholder sheet would be used, all requirements first get a value based on the opinion of all stakeholders. The requirements that have the highest calculated value will be implemented. Fig. 5 illustrates how the additional activities would be added to the PDD of the MOSCOW method and how the deliverables change. Changes are marked in grey.

As can be seen in Fig. 5, in Wiegers' Features Prioritization Matrix, the value of a requirement is calculated by estimating the benefits, penalties, costs and risks per requirement on a scale from 0-9 [21]. All requirements and corresponding values are stored in a spreadsheet. Wiegers developed the 'multiple stakeholder sheet', which is very useful in case there are multiple stakeholders that have different visions when it comes to the variables that result in the value of a requirement. In this case, those multiple stakeholders are the product manager and the developers. After assigning the values, the priorities are calculated and used as a basis for selecting the requirements for the next release.

4.2 Discussion

Using the multiple stakeholder sheet and thus the Features Prioritization Matrix seems be a useful way of integrating the opinion of all stakeholders, including partners, into the requirements prioritization method. The main advantage of adapting the method this way is that all involved stakeholders get an opportunity to influence the requirements prioritization. Additionally, the classification of requirements is turned into a calculated result out of estimating variables instead of an estimation of the importance of the overall requirement. This results most likely into a more accurate and realistic requirement prioritization.

The case study carried out is a first evaluation of the idea of incremental method engineering, through marching situational factors. Although often described in literature, not many practical examples have been presented. Therefore, we believe that although this is just a single case study, it is an important contribution to the method engineering field. However, in order to strengthen our argument, we should carry our more case studies [11]. Also, the method base in this research contained eight requirements prioritization methods. Further research can be done with a larger method base, in order to fine tune a method more specifically to the situational factors of a case company.

Furthermore, instead of using the current situational factors of the case company, it might also be interesting to use situational factors that the company predicts or aims to reach in the near future. For example, if a company wishes to expand its number employees this could be registered in the list of situational factors while matching them to a suitable method. By doing this the company might be less likely to outgrow its method in a short time.

A last important issue for further research is the evaluation of the method fragment implementation at the case company. Currently, we link this implementation to an increase in maturity. However, more interesting is whether the increment also leads to

an increase in performance. Indicators that could be used for this are duration of the decision process, customer satisfaction or time-to-market.

5 Conclusion

In this research we have analyzed the requirement prioritization method of a case company according to maturity level and situational factors. We have also analyzed eight requirement prioritization methods on maturity level and situational factors to find out which of these could be implemented into the case company's method in order to let this method evolve to a higher maturity level. We used a comparison with keys (candidate methods) and a keyhole (case company's method) to visualize how a suitable method can be chosen out of all candidate methods. By doing this we have answered our main research question and illustrated how incremental ME can support the maturing of an information systems development method in a product software company.

We have found that the case company implemented the MOSCOW requirement prioritization method, which contains maturity levels A-D. In order to mature the method, level E would need to be added. Three out of the eight methods we analyzed contained maturity level E. By comparing the situational factors of these three methods with the situational factors of the case company, we have found that one method (Wiegers' Features Prioritization Matrix) is suitable to add to the existing method in order to let it mature.

For further research, we plan to carry out more case studies and extend the method base with more method fragments. Furthermore, we aim to verify whether the proposed method increments actually improve performance.

References

1. van de Weerd, I., Versendaal, J., Brinkkemper, S.: A Product Software Knowledge Infrastructure for Situational Capability Maturation: Vision and Case Studies in Product Management. In: Proceedings of the 12th Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 97–112 (2006)
2. van de Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., Bijlsma, L.: Towards a Reference Framework for Software Product Management. In: 14th International Requirements Engineering Conference, Minneapolis/St. Paul, MN, USA, pp. 319–322 (2006)
3. Brinkkemper, S.: Method Engineering: Engineering of Information Systems Development Methods and Tools. Information and Software Technology 38(4), 275–280 (1996)
4. Kumar, K., Welke, R.J.: Methodology Engineering: A Proposal for Situation-Specific Methodology Construction. In: Challenges and Strategies for Research in Systems Development, pp. 257–269. John Wiley & Sons, Inc., New York (1992)
5. Tolvanen, J.-P.: Incremental Method Engineering with Modeling Tools: Theoretical Principles and Empirical Evidence. Jyväskylä Studies in Computer Science, Economics and Statistics 47, University of Jyväskylä, PhD Dissertation thesis (1998)
6. Aydin, M.N., Harmsen, F.: Making a method work for a project situation in the context of CMM. In: Oivo, M., Komi-Sirviö, S. (eds.) PROFES 2002. LNCS, vol. 2559, pp. 158–171. Springer, Heidelberg (2002)

7. Ralyté, J., Deneckère, R., Rolland, C.: Towards a Generic Model for Situational Method Engineering. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 95–110. Springer, Heidelberg (2003)
8. Ågerfalk, P.J., Wistrand, K., Karlsson, F., Börjesson, G., Elmberg, M., Möller, K.: Flexible Processes and Method Configuration: Outline of a Joint Industry-Academia Research Project. In: 5th International Conference on Enterprise Information Systems (2003)
9. Rossi, M., Ramesh, B., Lyytinen, K., Tolvanen, J.: Managing Evolutionary Method Engineering by Method Rationale. Journal of the Association for Information Systems 5(9), Article 12 (2004)
10. Bekkers, W., van de Weerd, I., Spruit, M., Brinkkemper, S.: A Framework for Process Improvement in Software Product Management. In: Riel, A., O'Connor, R., Tichkiewitch, S., Messnarz, R. (eds.) EuroSPI 2010. CCIS, vol. 99, pp. 1–12. Springer, Heidelberg (2010)
11. Yin, R.K.: Case Study Research: Design and Methods, 4th edn. SAGE Publications, California (2009)
12. Bekkers, W., van de Weerd, I., Brinkkemper, S., Mahieu, A.: The Influence of Situational Factors in Software Product Management: An Empirical Study. In: 2nd International Workshop on Software Product Management, pp. 41–48 (2008)
13. van de Weerd, I., Brinkkemper, S.: Meta-modeling for Situational Analysis and Design Methods. In: Syed, M.R., Syed, S.N. (eds.) Handbook of Research on Modern Systems Analysis and Design Technologies and Applications, pp. 38–58. Idea Group Publishing, Hershey (2008)
14. Bebensee, T., van de Weerd, I., Brinkkemper, S.: Binary Priority List for Prioritizing Software Requirements. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182, Springer, Heidelberg (2010)
15. Boehm, B.: A Spiral Model of Software Development and Enhancement. Computer, 61–72 (May 1988)
16. van den Akker, M., Brinkkemper, S., Diepen, G., Versendaal, J.: Software Product Release Planning Through Optimization and What-if Analysis. Information and Software Technology 50(1-2), 101–111 (2008)
17. Davis, A.M.: The Art of Requirements Triage. Computer 36(3), 42–49 (2003)
18. Stapleton, J.: DSDM Business Focused Development. Addison-Wesley Professional, Reading (2002)
19. Karlsson, J., Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. IEEE Software 14(5), 67–74 (1997)
20. Mizuno, S., Akao, Y. (eds.): Quality Function Deployment: Integrating Customer Requirements into Product Design. Productivity Press Inc., Portland (1990)
21. Wiegers, K.E.: First Things First: Prioritizing Requirements. Software Development Magazine, 24–30 (September 1999)
22. Stapleton, J.: Dynamic Systems Development Method. In: Proceedings of the Technology of Object-Oriented Languages and Systems, June 07-10, p. 406 (1999)

Design Solution Analysis for the Construction of Situational Design Methods

Robert Winter

Institute of Information Management, University of St. Gallen
Müller-Friedberg-Strasse 8, 9000 St. Gallen, Switzerland
Robert.Winter@unisg.ch

Abstract. Situational design methods provide problem solving guidance that can be configured to fit a range of different design goals and contexts. While the formal aspects of situational method engineering are well researched, the specification of method fragment instances and their configurations is often left open and regarded as specific to the respective design problem class. We propose an approach that analyzes variations of existing design solutions to explore the underlying design factors and to identify design situations. This knowledge is then used to derive method fragments and configuration rules that represent the explored variety of design solutions. The proposed approach has been applied for several design problem classes to construct concrete situational design methods. As an illustrative example, the construction of a situational design method for enterprise architecture management is used in this paper. Based on the exploration of eight design factors and three design situations, six method fragments are derived that are combined into four situational method configurations.

1 Introduction: Contingencies, Design and Situational Methods

"At the most abstract level, the contingency approach says that the effect of one variable on another depends upon some third variable" [1]. In an organisational context, contingency theory argues that success or failure of different organizational structures depends on contingency factors such as size, task uncertainty, and task interdependence [2]. Although its validity is questioned by some [e.g. 3], we agree with Donaldson that "overall, empirical studies show that fit positively affects performance, thereby supporting the central idea of contingency theory" [1].

Design Science Research is a research paradigm that has been, among other application domains, successfully deployed to Information Systems (IS). In the following, we will use DSR to abbreviate Design Science Research in Information Systems. At its core, DSR is about the rigorous construction of ***useful*** IS artefacts, i.e. "technology-based solutions to important and relevant business problems." [4, table 1] Technically, IS artefacts can be constructs, models, methods, or instantiations [5]. While instantiations are usually represent a solution to a singular design problem, constructs, models and methods can have different levels of ***generality*** and, as a consequence, "represent [.] a general solution to a class of [design] problems." [6]

Generality is therefore an important quality of an IS artefact [4]. The two design goals of generality and utility are however conflicting. In their research on reference

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 19–33, 2011.

modelling, Becker et al. discuss what they call the [process] ***reference modelling dilemma***: "On the one hand, customers will choose a [process] reference model that […] provides the best fit to their individual requirements and therefore implies the least need for changes. On the other hand, a restriction of the generality of the model results in higher turn-over risks because of smaller sales markets" [7]. This dilemma is not only apparent in [process] reference modelling, but also exists for design methods. With increasing generality, the individual utility of a solution for solving a specific design problem decreases – and vice versa. The overall, cross-organization sum of individual utilities might be increasing when design solutions have a higher generality – but individual organizations might not be interested in this "overall utility". As a solution to this dilemma, Becker et al. [7] propose adaptation mechanisms that instantiate a generic [process] reference model according to the specific design problem at hand.

Both design methods and [process] reference models can be understood as (more or less) general problem solutions. As a consequence, situational methods (e.g. [8, 9, 10, 11]) can, like adaptable [process] reference models, aim at solving the trade-off between solution generality and individual solution utility.

As situational method engineering allows to develop artefacts which are adaptable to different design problem instances within a design problem class, a crucial decision during the method construction phase is to delineate the range of addressed design problems (i.e. to specify the design problem class) and to understand the relevant ***design situations*** within this class. If a design problem class is understood as a set of "similar" design problems, a design situation can be understood as a subset of design problems which are even more similar, i.e. which share certain contingencies. It has been argued that, in situational method engineering, such contingencies can be represented by a certain design goal vector and/or by a certain context [12]. Depending on the degree of desired generality, a design problem class can be partitioned into few, very generic design situations or a larger number of design situations of lesser generality.

The configuration or adaptation of a situational method to a certain design situation can therefore be understood as an application of contingency theory. If relevant contingencies of the respective design problem class are represented correctly by appropriate design situations and appropriate *adaptation/configuration mechanisms*, design solutions can be generated that not only solve the [process] reference modelling dilemma, but also consider contingency factors. To achieve this, however, method engineering must identify the contingencies of a design problem class and correctly derive not only a set of suitable design situations, but also adaptation/configuration mechanisms that combine method fragments into situational methods.

The aim of this paper is to show that method construction and contingency identification can be based on an analysis of a sufficient number of existing design solutions for the addressed design problem class. In section 2 we summarize and extend an existing design solution analysis approach. Section 3 outlines how design solution analysis is used to derive method fragments and fragment configuration rules. As an illustrative example, enterprise architecture management is used as the design problem class, and a respective situational method is derived in section 4. Section 5 summarizes the paper and outlines future research in this domain.

2 Analysis of Existing Design Solutions

Winter [13] extended Bucher and Klesse's design problem analysis procedure [14] by differentiating more components and assuming that, in general, only adaptable, situational solution artefacts are constructed. In the following, Winter's proposal is refined and illustrated:

1. A ***rough idea*** about the delineation of the design problem class is developed. Results of this step are definitions, a description of the system under analysis and ideas about design goals for the respective class of design problems.

   In section 3 we illustrate our approach for the design problem class Enterprise Architecture Management (EAM). It is delineated by defining architecture, enterprise architecture and EAM, by defining the scope of relevant artefacts, and by differentiating potential EAM goals.
2. A ***literature analysis*** is conducted in order to identify potential contingency factors for the respective class of design problems, i.e. factors which might have influence on how such design problems are solved in practice.

   For EAM, such an analysis yields factors like 'EAM's main sponsor is IT or business', 'EAM's main deliverable is maps, analyses or project support', 'EAM's main goal is transparency, consistency, simplification, or flexibility', or 'EAM's role is active or passive'.
3. A ***field study*** is conducted in order to analyze how design solutions for this class of design problems in practice are actually related to what contingencies. Using ***principal component analysis*** on the field study data, the list of potential contingency factor candidates from step 2 is reduced to a smaller set of relevant "design factors". Design factors are usually aggregates of several relevant contingency factors and therefore need to be semantically interpreted.

   For EAM, principle component analysis on EAM practice solutions yielded eight design factors (like IT operations support, integrative role, business strategy support, or design impact) which aggregate 54 statistically relevant contingencies (see section 4). E.g., the design factor 'integrative role of EAM' aggregates the contingencies 'EAM takes place in an interdisciplinary team', 'EAM team and business departments continuously exchange information (e.g. in architecture boards)' and 'EAM team and IT departments continuously exchange information (e.g. in architecture boards)'.
4. In a multi-dimensional room where every dimension corresponds to a design factor, every observed solution can be understood as a point. The design problem class now should be ***redefined*** by specifying value ranges for the design factors identified in step 3. This means that "outlier" design solutions are excluded from further analysis in order to ensure a useful degree of homogeneity of solutions.

   For EAM step 4 leads to the exclusion of few observed solutions which can be clearly recognized as outliers, i.e. which clearly cannot be associated with any cluster in the above described multi-dimensional room.
5. For the vast majority of observations, ***ultrametric distances*** can now be computed that represent the similarity (or dissimilarity) between design solutions. Metrics for ultrametric distances are usually based on Euclidian distance. The observations and their distances can be visualized using a dendrogram-like tree graph. The

(dis)similarity of two design solutions corresponds to the generality level of their link. If two design solutions are very similar, their link is represented on a low level of generality. If two design solutions are very different, their link is represented on a very high level of generality. The linkage can be interpreted as generalization of the linked specific solutions.

Figure 1 [adapted from 13] illustrates such a tree graph that represents 33 observed solutions (C1...C33, on the bottom of the tree diagram) in design problem class C as well as their ultrametric distances. The generalization of solutions C11, C12, C13, C14 and C15 is represented as generic solution C11...C15 on some level of generality. The generalization of solutions C1 through C15 is represented as even more generic solution C1...C15 on a higher level of generality. The maximum generic solution of design problem class C is found at the top of the tree diagram.

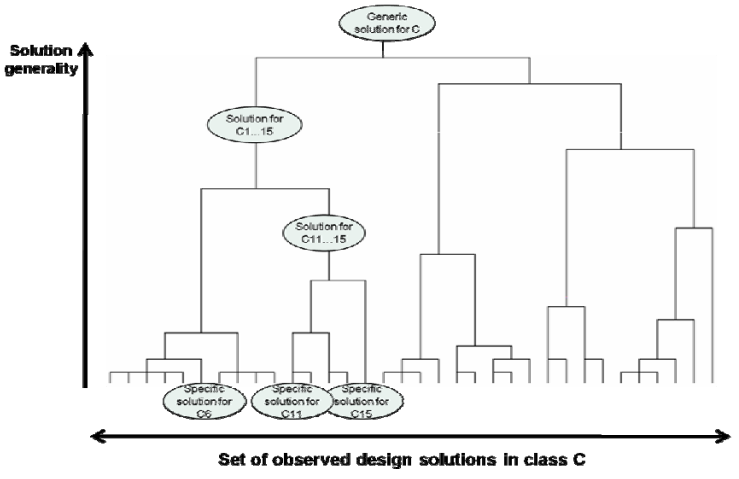


Fig. 1. Ultrametric tree visualization of observed design solutions for design problem class C

6. In order to not only visualize, but characterize generic solutions in C, a ***clustering algorithm*** can be applied to the observation data. By agglomerative clustering, solutions can be specified at any generality level between "full detail" (i.e. one cluster per original observation) and "one size fits all" (i.e. one generic solution description for the entire design problem class). Analyzing the clustering error in relation to the number of clusters, an optimal level of generality (i.e. an optimal number of clusters) can be determined.

   Empirical data can be used to determine the optimal number of clusters, i.e. the number of different 'approaches' that should be differentiated, e.g. for a certain number of companies that implemented EAM. If the set of observations is large and diverse enough, this finding might be applied to EAM in general.
7. For the level of solution description generality chosen in step 6, each cluster represents one ***design situation***. The situations should not only be defined formally (i.e. by specifying value ranges of the respective design factors), but also should be interpreted semantically ("design problem types").

EAM clusters differ in particular with regard to their values for the design factors IT operations support, integrative role, design impact, enterprise-wide focus and IT strategy support. These differences are used to characterize one cluster as 'balanced, active EAM', one as 'business analysis' and one as 'IT focused, passive EAM'. As a consequence, situational method engineering should provide three situated methods. Since the clusters are similar with regard to some (but not all!) design factors, these situated methods will share certain method fragments.

There is some, but not much, related work on how to identify design situations for situational method engineering [15]: Some suggest to differentiate "project size", "number of stakeholder groups" or "applied technology" for every situational method (e.g. [16] and [17] according to [18]), while others specify situations on a case-by-case basis [e.g. 10, 12]. The procedure proposed here allows systematic and reliable identification and specification of design situations for any given class of design problems as long as a sufficient number of problem solutions can be observed and analyzed for this problem class.

But identifying design factors and design situations is only the first part of constructing a situational design method. Design problems need to be identified and linked to design situations, and a set of method fragments needs to be specified whose combinations will constitute useful solutions for such design problems. The next section proposes a procedure for this second part of situational design method construction.

It should however be noted that even both construction procedure parts are not necessarily sufficient to solve every design problem in C. Situated methods might need to be adapted to provide a useful design solution to a specific design problem. Referring to fig. 1, a combination of method fragments might solve the generic design problem for situation C1..15 sufficiently, but still might need to be adapted to design problem C15 in order to solve this specific problem instance effectively.

3 Derivation of Method Fragments and Configuration Rules

For typical design problem classes, between four and eight design factors can be identified which explain the variance of the observed design solutions sufficiently [19, 20, 21, 22, 23, 24]. These design factors span up a solution room where between three and six design situations are differentiated.

The crucial step is to qualitatively interpret the n design factors and m design situations that have been quantitatively created as principal components of the data set and clusters in the n-dimensional design factor space, respectively. For that purpose, it is necessary for every design situation to identify the subset of p design factors ($p \leq n$) that best characterizes the respective design situation, i.e. whose factor values are particularly high or low in a cluster and/or whose factor values have only a small standard deviation in a cluster.

Understanding the problem-oriented relations between design factors and design situations is essential for the construction of respective methods: Each method fragment can then be interpreted as an "***elementary movement***" in the p-dimensional design factor sub-space. The situated method aggregates certain fragments and therefore constitutes a complex, multi-dimensional movement in the n-dimensional design factor space.

8. For every design situation ***characterizing design factors*** need to be identified. In EAM, only the design situation 'IT focused, passive EAM' is characterized by high values of the design factor IT operations support and low values of the design factors enterprise-wide focus, integrative role and design impact. The EAM design situation 'balanced, active EAM', in contrast, exhibits much smaller values for IT operations support, but much higher values for enterprise-wide focus, integrative role and design impact. With regard to information supply, business support and IT strategy and IT governance support, these two design situations are not very different, so that these factors are not useful to characterize them.
9. Now characterizing design factors need to be linked to ***design problems***. All preceding procedure steps analyze existing design solutions. Since these design solutions were created purposefully, they are qualitatively interpreted and linked to design problems. For 'IT focused, passive EAM', the characterizing design factors 'integrative role', 'enterprise-wide focus' and 'design impact' can be associated with an EAM setup where the main EAM sponsor is the CIO, the main EAM customer is the IT function, EAM is primarily performed within the IT function and EAM is widely ignored by business units. 'Business analysis', in contrast, can be associated with an EAM setup where business is the main stakeholder and executor and where implementation considerations are widely neglected. Most EAM setups can be easily linked to major EAM challenges as often described in the literature. E.g., missing business involvement and missing business value creation of EAM correspond to the first EAM setup, while missing 'grounding'/'execution' and too much 'locality' of EAM correspond to the latter EAM setup.
10. Elementary problem-solving actions are now derived by comparing design solutions (steps 1-8) with design problems (step 9). These elementary problem-solving actions constitute ***method fragments***. If e.g. the design problem is that business stakeholders are not sufficiently involved in EAM sponsorship and/or EAM delivery, and that EAM recommendations seem not to create sufficient business value, the as-is EAM setup is close to 'IT focused, passive EAM' while the to-be EAM setup is likely to be 'Balanced, active EAM'. Elementary problem-solving activities can be derived from the respective characterizing design factors 'integrative role', 'enterprise-wide focus' and 'design impact'. A suitable method fragment should include, among others, 'EAM alignment with business goals', 'architects have an extensive network within the company', 'EAM team and business departments continuously exchange information (e.g. in architecture boards)', 'EAM takes place in an interdisciplinary team' and 'EAM has an impact on business architecture design'. If, as another example, the design problem is that EAM is not sufficiently 'implemented' and creates not enough impact, the as-is EAM setup is close to 'business analysis' while the to-be EAM setup is likely also to be 'Balanced, active EAM'. Elementary problem-solving activities can be derived from respective characterizing design factors 'IT governance and IT strategy support', 'IT operations support' and 'EAM governance'. Hence a suitable method fragment should include, among others, 'Results of EAM are used for IT strategy development', 'Architecture data is centralized with the EAM department' and 'Results of EAM are used for IT development'.
11. Based on the set of identified design problems and specified method fragments, now ***method configuration rules*** need to be derived. Basically the (reusable)

method fragments identified in step 10 need to be related to respective design situations. The fewer characterizing design factors and the fewer design problems have been identified, the simpler the fragment configuration will be – and vice versa. For the EAM example, four situated methods are configured from, depending on the design situation, up to four method fragments out of a total number of six reusable method fragments (see section 4).

4 Enterprise Architecture Management – An Illustrative Example for Design Solution Analysis

The ANSI/IEEE Standard 1471-2000 defines architecture as "the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution" [25]. For enterprise architecture, relevant system views are strategic positioning, organizational structure, process organization, information flows, and implementation by means of software systems and data structures [26]. EAM can provide systematic support to organizational change that affects business structures as well as IT structures by providing constructional principles for designing the enterprise [27]. The development and evolution of EAM need to be based on appropriate design methods. EAM methods typically comprise strategic design of an architectural vision, development and maintenance of as-is architecture models, development and maintenance of to-be architecture models, migration planning, implementation of enterprise architecture, and analysis of enterprise architecture on the basis of architecture models [28].

Aiming at a deeper understanding of the constituent factors that influence EAM, there has been some scientific effort to analyze contingency factors of EAM. Aier et al. [29] have identified models, data, and organizational penetration as potential contingencies. They did however not explicitly consider management aspects of EAM. Leppänen et al. [30] made a first step towards a complex contingency framework for an engineering method for enterprise architecture. Ylimäki [31] conducted several studies in order to identify potential critical success factors for EAM, yielding commitment, governance, methodology, enterprise architecture models, project management, training and education, organizational culture, IT investment strategy, assessment and evaluation, business-driven approach, communication, and scope. These success factors give a first insight into possible design factors of EAM.

4.1 Procedure Steps 1 through 7: Design Solution Analysis

This subsection summarizes Aier et al.'s empirical analysis of EAM design solutions [32] that applied steps 1 through 7 of the proposed procedure – although not elaborating the procedure itself. Aier et al. use Ylimäki's set of EAM success factors as a starting point. In order to distinguish different EAM approaches, the first part of their questionnaire asks for a company's general EAM understanding. Then, the EAM positioning is analyzed using questions on EAM integration in the organization and on the way how organizational units, teams and roles are involved in EAM processes. Other important aspects in this context are the scope of EAM processes, the penetration of EAM processes / EAM results throughout the organization as well as the level

of continuity and controlling of EAM processes. In the third and final part of the questionnaire, it is asked what types of EAM results are used by which organizational units. All in all, 54 questions are used to assess current EAM design solutions in companies. At four EAM practitioner events, a total of 119 data sets were collected that did not reveal substantial extent of missing data (10% at maximum).

In order to identify design factors for EAM, Aier et al. apply an exploratory factor analysis. The original study [32] documents two quantitative techniques that support the suitability of the data set for Principal Component Analysis. Using Varimax rotation with Kaiser normalization, eight design factors are identified that comprise a total of 38 questionnaire items. 16 questionnaire items were deleted because they were intentionally designed as control items or did not seem to contribute to the factor identification [33]. Due to some incomplete questionnaires, missing values were excluded pair wise during the factor analysis. This resulted in a total number of 109 cases contributing to the factor analysis. The items selected for the factor analysis explain 67.63% of the variance in total. The original study [32] also documents a quantitative technique that supports the validity of the factor analysis.

With regards to the interpretation of the factors, factor loadings from 0.3 to 0.4 are considered to be the minimal level [33], while factor loadings from at least 0.5 are considered as sufficient for an unambiguous assignment to one factor. For some items that showed identical factor loadings for more than one factor, the factor was chosen that best matched the respective factor from an EAM literature perspective.

The study yields the following eight EAM design factors:

a) ***IT operations support:*** The use of results for IT operation tasks and by IT departments for their daily job characterizes this factor. Considering the items' loadings on this factor it becomes obvious that usage of EAM results as well as the perception of EAM within the organizational units concerned with IT operations exert a conjoint effect on the overall assessment of EAM.
b) ***Support of management tasks by EAM:*** This is again expressed by the usage of EAM results by management tasks as well as by the perception of EAM in the management board. This factor constitutes the "antipole" to factor (a) and reveals that EAM can serve both IT and management purposes, but that these purposes are most probably not highly interrelated. It can be assumed that a high degree of realization for factors (a) and (b) might distinguish different EAM approaches fundamentally.
c) ***Governance of EAM:*** EAM governance consists of model and process assessment and maintenance and a central supervision of EA models and data.
d) ***Support of IT strategy and IT governance tasks:*** EAM and its results are considered to be an essential part of IT strategy development and IT governance.
e) ***Information supply:*** This design factor reflects the service function that EAM can fulfil both for business and IT departments. Moreover the support of business/IT alignment is an essential part of this factor.
f) ***Integrative role:*** The integrative role of EAM can be realized by interdisciplinary teams and a continuous exchange between EAM roles. It can be assumed that the existence of an architecture board is part of such an organizational structure for EAM.
g) ***Design impact:*** EAM can impact IT or infrastructure, application or business architecture. The degree of design impact reflects the penetration of the EAM approach throughout the organization as well as its active role.

h) ***Business strategy support:*** In contrast to design factor (b), items in this design factor describe the support of strategic tasks that are not management tasks - like e.g. enterprise development and product planning. Most probably, high degrees of realization of this factor correspond to a high realization of design factor (b).

Three different groups of EAM design factors were found: Design factors (a), (b), (d), (e) and (h) characterize the ***concern*** of the EAM approach (i.e. if the approach supports IT operations, management tasks, IT strategy, Business/IT alignment or business strategy). Design factors (f) and (g) describe the ***role*** of the EAM approach within the company (moderator or innovator). Finally, design factor (c) describes the ***governance*** of the EAM approach itself.

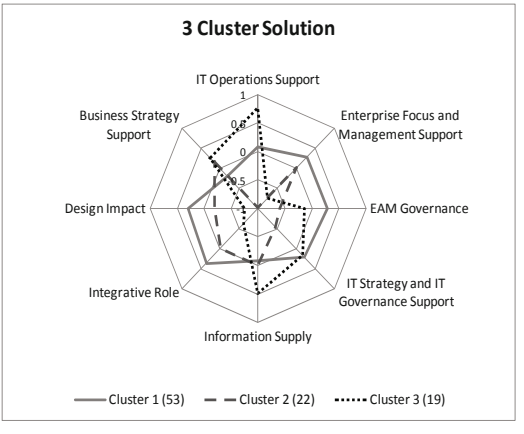


Fig. 2. EAM design situations (and their representation in the data set) [32]

Considering results from preliminary cluster analyses on the data, one case was eliminated as it showed significant outlier behaviour [33]. Excluding cases with missing factor loadings, 94 cases were used for the cluster analysis. Aier et al [32] applied the *Average Within-Group Linkage* cluster algorithm provided by SPSS and *Squared Euclidean Distance* as the distance measure. They identified three EAM design solution clusters that can be regarded as EAM design situations.

In fig. 2, the three identified EAM design situations are illustrated as a cobweb diagram. The eight EAM design factors are used as dimensions; Each cluster's centroid value was used to represent the respective cluster's values. The cobweb diagram nicely visualizes the characteristics of each EAM design situations.
The EAM design situations can be described as follows: [32]

- ***Design situation 1: Balanced, active EAM:*** The first cluster (solid line in fig. 2) presents a rather balanced approach to EAM. For most factors this cluster shows the highest or at least average values. Especially the similar values for the factors IT operations support and enterprise-wide focus lead to the conclusion that organizations within this cluster focus neither on IT support nor on management support. In contrast to the other clusters, the high support of IT operations, management, IT strategy as well as the focus on design impact, the integrative role and EAM

governance argue for a high degree of integration within the organization. In particular the values for design impact, integrative role and EAM governance are by far the highest between all three clusters. It can therefore be presumed that these organizations have a rather high level of maturity in their EAM approach.

It should be noted that this cluster includes 53 out of 94 organizations, which lead to the supposition that this cluster represents a "mainstream" approach.

- ***Design situation 2: Business analysis:*** The second cluster (dashed line in fig. 2) groups 22 organizations that have an apparent focus on business support in their EAM approach. The factors IT operations support as well as IT strategy and IT governance support are clearly assigned with comparatively low values. Comparing the mean factor values to those of cluster 1, the overall low values imply that the organizations in this cluster do not show a high degree of EAM implementation in any dimension. Two conclusions can be derived from this fact: Firstly, the organizations could have decided to apply a minimalist EAM approach, focusing on management support without putting resources in EAM governance or an active role of EAM. Second, the introduction of EAM could only recently be initiated by management and is not very mature yet. For both cases, literature suggests that a sustainable EAM approach can only be established by realizing an effective EAM governance [34, 35].
- ***Design situation 3: IT focused, passive approach:*** Organizations assigned to this cluster (dotted line in fig. 2) clearly emphasize the use of EAM for IT operations as well as the information supply by EAM. In contrast, values for management support are by far the lowest compared to the other clusters. As the factors design impact as well as integrative role are not focused by this approach, it can be described as a passive approach that is most probably realized very locally within the organization.

  Obviously, this small cluster, which includes only 19 of the 94 organizations, represents a specialized IT-centred EAM approach that primarily takes a documentation role. It can be presumed that the EAM approach was initiated by IT departments and has not been disseminated throughout the organization yet.

After delineating the design problem class, collecting empirical design solution data, identifying and interpreting design factors, and identifying and interpreting design solution clusters, the 'analysis' portion of the proposed procedure is completed. The new steps 8 through 11 address the construction of a situational design method on that basis.

4.2 Procedure Steps 8 through 11: Design Method Construction

The description of design situation 1 shows that this situation rather constitutes a mature, to-be state rather than a design problem. Compared to situation 1, situations 2 and 3 exhibit clear gaps and therefore can be considered as EAM design problem subclasses. Companies that have not systematically realized EAM at all will however not be included in any of the clusters so that, in addition to the above two sub-classes, an EAM design method needs to address a third problem sub-class. If we assume that no direct transformation from "no systematic EAM at all" to the quite mature state in situation 1 is feasible, two alternatives of this third sub-class have to be regarded: from nothing to situation 2 vs. from nothing to situation 3.

In the following, we characterize each design problem sub-class (not design situation!) by assigning characterizing design factors, and we derive design method fragments from that assignment:

- ***Design problem sub-class I (from situation 2 to situation 1):*** IT operations support, IT strategy and IT governance support as well as EAM governance need to be strengthened. Since IT topics and EAM governance constitute widely different measures, two method fragments (designated as A and B) should be differentiated to achieve that transformation.
- ***Design problem sub-class II (from situation 3 to situation 1):*** Design impact, integrative role and enterprise-wide focus need to be strengthened. Since design impact and IT/business alignment issues constitute widely different measures, two additional method fragments (designated as C and D) should be differentiated to achieve that transformation.
- ***Design problem sub-class III (from nothing to situation 3):*** If an IT focused approach is favoured, IT operations support, IT strategy and IT governance support, information supply and business strategy support need to be developed foremost. In addition to fragment A (see above: IT operations, IT strategy and IT governance support), two additional fragments should be defined: fragment E to address business strategy support and fragment F to address information supply.
- ***Design problem sub-class IV (from nothing to situation 2):*** If a business analysis approach is favoured, enterprise-wide focus as well as information supply and business strategy support need to be developed foremost. Such a transformation is represented by the fragments D, E and F.

Based on EAM design factors (a)...(h) and EAM design situations (1)...(4), EAM design problem sub-classes (I)...(IV) have been derived that can be addressed by different configurations of EAM method fragments (A)...(F). It should be noted that (A)...(F), although designated as fragments here due to their configuration into situational methods, are quite complex and would be designated as method components or even methods in a different context (e.g. D as a method for EAM-based IT/business alignment). As a result from applying steps 8 through 11 of the proposed procedure, the following situational EAM method is constructed:

- ***Method for problem sub-class I (from business analysis to balanced, active EAM):*** combine method fragments A and B
- ***Method for problem sub-class II (from IT focused, passive to balanced, active EAM):*** combine method fragments C and D
- ***Method for problem sub-class III (initial development of IT focused, passive EAM):*** combine method fragments A, E and F
- ***Method for problem sub-class IV (initial development of business analysis):*** combine method fragments D, E and F

Although not advised because a big maturity leap is necessary, it is possible to combine method fragments A, D, E and F into a method for initial development of a balanced, active EAM.

5 Conclusions and Outlook

While many method engineering approaches claim to incorporate situational factors, they do nearly never detail what these situational factors exactly are and how they can incorporated into method fragment design and fragment configuration rules. This paper is based on Winter's analysis procedure [13] that has been applied to EAM design solution analysis in [32]. Since earlier extensions of the proposed analysis procedure for constructing situational methods suffer from too simplistic design situations and design problems [cf. 36], we have used here the EAM data that allows a more realistic illustration of the proposed procedure extensions. The proposed procedure guides not only the identification of relevant context and project type factors, examines their occurrences in practice, and classifies them into design solution situations. Based on an evaluation of solution maturity, design problems can be specified and associated with matching design factors. From that association, transformation fragments and their configuration into situational methods (one for each design problem sub-class) can be derived.

The question arises how general the proposed situational method engineering procedure is. While its construction portion has only been applied to IT/business alignment [36] and EAM so far, its analysis portion has been applied in many cases:

- Leist [23] uses it to identify eight enterprise modelling situations. Based on that analysis, she investigates which meta modelling approaches are best suited in which situation.
- Baumöl [19] uses it to identify four types of transformation projects. Based on that analysis, she constructs project type specific recommendations which general and type specific transformation management instruments should be used.
- Bucher and Winter [20] use it to identify four Business Process Management (BPM) realization situations and five types of BPM transformations. Based on that design problem analysis, they construct a situational BPM method.
- Klesse and Winter [21] use it to identify four organizational designs for data warehouse service providers. Based on that analysis, they give recommendations for consistent data warehousing service provisioning and identify dynamic patterns (maturity).
- Lahrmann and Stroh [22] use it to identify three approaches to organize and implement information logistics in companies. Based on that analysis, they derive guidelines and reference models for information logistics strategy design.
- Aier, Gleichauf, Riege and Saat [24] use it to verify a hypothesis that all integration projects in companies can be assigned to one of only four fundamental types.

Although the generality of the proposal needs to be demonstrated yet, there is some evidence that the procedure in general can be applied to a wide variety of IS related design problems in organizations.

Four broad categories of research opportunities exist: Firstly, the demonstration example lacks tradeoffs between design goals and design activities: Since implementation impact and business involvement should be achieved equally, the derivation of fragments and their aggregation into situated methods therefore was straightforward. If tradeoffs have to be observed, both the fragment specification and the fragment

aggregation become much more complex. Secondly, an interesting feature of many design solution analyses that yield a larger number of design factors is that the first factor is often representing many and quite diverse problem aspects that are sometimes not easy to interpret qualitatively. With regard to design solution analysis and method construction, we interpret this "technically" overloaded design factor as a problem independent aggregation of "generalized" properties and the respective solution fragment as a basic set of domain-independent problem solution activities like e.g. general project/transformation management. This aspect of our approach does certainly need additional research attention. Thirdly, the proposed approach does not explicitly cover yet the adaptation of situated methods to specific design problems. On the one hand, we consider this extension not too problematic because there is a plethora of adaptation knowledge on reference models which promises to be generalizable. On the other hand, adaptation efforts might depend on problem properties and influence the "optimal" level of method generality that up to now is determined using "technical" homogeneity/heterogeneity metrics only. Finally, another and probably the most important extension of the proposed approach would be the inclusion of not only adaptation effort, but also other "economical" properties like the absolute number of design problems in a class or even their attractiveness in terms of economic gains. This is probably the most interesting – and challenging – avenue for further research.

References

1. Donaldson, L.: The Contingency Theory of Organizations. Sage, Thousand Oaks (2001)
2. Graubner, M.: Task, firm size, and organizational structure in management consulting. An empirical analysis from a contingency perspective. DUV, Wiesbaden (2006)
3. Pfeffer, J.: New Directions for Organization Theory: Problems and Prospects. Oxford University Press, New York (1997)
4. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Quarterly 28(1), 75–105 (2004)
5. March, S.T., Smith, G.F.: Design and Natural Science Research on Information Technology. Decision Support Systems 15(4), 251–266 (1995)
6. Baskerville, R.L., Pries-Heje, J., Venable, J.: Soft design science methodology. In: Proceedings of DESRIST 2009. ACM, New York (2009)
7. Becker, J., Delfmann, P., Knackstedt, R.: Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In: Becker, J., Delfmann, P. (eds.) Reference Modeling, pp. 27–58. Physica, Heidelberg (2007)
8. Brinkkemper, S., Saeki, M., Harmsen, A.F.: Meta-Modelling Based Assembly Techniques for Situational Method Engineering. Information Systems 24(3), 209–228 (1999)
9. Harmsen, A.F., Brinkkemper, S., Oei, H.: Situational Method Engineering for Information System Project Approaches. In: Proceedings of the IFIP 8.1 Working Conference on Methods and Associated Tools for the Information Systems Life Cycle, pp. 169–194. North-Holland, Maastricht (1994)
10. Mirbel, I., Ralyté, J.: Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering 11(1), 58–78 (2006)
11. Ralyté, J., Rolland, C.: An approach for method reengineering. In: Kunii, H.S., Jajodia, S., Sølvberg, A. (eds.) ER 2001. LNCS, vol. 2224, pp. 471–484. Springer, Heidelberg (2001)

12. Bucher, T., Klesse, M., Kurpjuweit, S., Winter, R.: Situational Method Engineering – On the Differentiation of "Context" and "Project Type". In: IFIP WG8.1 Working Conference on Situational Method Engineering – Fundamentals and Experiences (ME 2007), pp. 33–48. Springer, Berlin (2007)
13. Winter, R.: Problem Analysis for Situational Artefact Construction in Information Systems (2011) (forthcoming)
14. Bucher, T., Klesse, M.: Contextual Method Engineering, Research Report BE HSG/EIW/03, University of St. Gallen, Institute of Information Management (2006)
15. Winter, R., Gericke, A., Bucher, T.: Method versus Model – Two Sides of the Same Coin? In: Dietz, J., Albani, A. (eds.) Advances in Enterprise Engineering II, Amsterdam (2009)
16. Kornyshova, E., Deneckère, R., Salinesi, C.: Method Chunks Selection by Multicriteria Techniques: an Extension of the Assembly-based Approach. In: IFIP WG8.1 Working Conference on Situational Method Engineering – Fundamentals and Experiences (ME 2007), pp. 64–78. Springer, Berlin (2007)
17. van Slooten, K., Hodes, B.: Characterizing IS Development Projects. In: Proceedings of the IFIP TC8, WG8.1/8.2 Working Conference on Method Engineering, pp. 29–44. Chapman & Hall, Atlanta (1996)
18. Rolland, C.: A Primer for Method Engineering, in Proceedings of the Informatique des Organisations d'Information et de Décision (INFORSID), Toulouse (1997)
19. Baumöl, U.: Strategic Agility through Situational Method Construction. In: Proceedings of the European Academy of Management Annual Conference (2005)
20. Bucher, T., Winter, R.: Taxonomy of Business Process Management Approaches: An Empirical Foundation for the Engineering of Situational Methods to Support BPM. In: vom Brocke, J., Rosemann, M. (eds.) Handbook on Business Process Management, Springer, Heidelberg (2010)
21. Klesse, M., Winter, R.: Organizational Forms of Data Warehousing: An Explorative Analysis. In: Proceedings of the 40th Hawaii International Conference on System Sciences (HICSS-40), IEEE Computer Society, Los Alamitos (2007)
22. Lahrmann, G., Stroh, F.: Towards a Classification of Information Logistics Scenarios - An Exploratory Analysis. In: Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS-42), IEEE Computer Society, Los Alamitos (2009)
23. Leist, S.: Methoden zur Unternehmensmodellierung - Vergleich, Anwendungen und Diskussionen der Integrationspotenziale, Habilitation, Institut für Wirtschaftsinformatik Universität St. Gallen (2004)
24. Aier, S., Gleichauf, B., Riege, C., Saat, J.: Empirische Validierung von Integrationstypen am Beispiel unternehmensübergreifender Integration. In: Proceedings der 9. Internationalen Tagung Wirtschaftsinformatik, Band 1, Wien, Österreichische Computer Gesellschaft, pp. 99–108 (2009)
25. IEEE: IEEE Recommended Practice for Architectural Description of Software Intensive Systems (IEEE Std 1471-2000), IEEE Computer Society, New York, NY (2000)
26. Winter, R., Fischer, R.: Essential Layers, Artifacts, and Dependencies of Enterprise Architecture. Journal of Enterprise Architecture 3(2), 7–18 (2007)
27. Dietz, J.L.G.: Enterprise Ontology – Theory and Methodology. Springer, Heidelberg (2006)
28. Aier, S., Riege, C., Winter, R.: Unternehmensarchitektur – Literaturüberblick und Stand der Praxis. Wirtschaftsinformatik 50(4), 292–304 (2008)
29. Aier, S., Riege, C., Winter, R.: Classification of Enterprise Architecture Scenarios – An Exploratory Analysis. Enterprise Modelling and Information Systems Architectures 3(1), 14–23 (2008)

30. Leppänen, M., Valtonen, K., Pulkkinen, M.: Towards a Contingency Framework for Engineering an Enterprise Architecture Planning Method. In: Proceedings of 30th Information Systems Research Seminar in Scandinavia, IRIS 2007 (2007)
31. Ylimäki, T.: Potential Critical Success Factors for Enterprise Architecture. Journal of Enterprise Architecture 2(4), 29–40 (2006)
32. Aier, S., Gleichauf, B., Winter, R.: Understanding Enterprise Architecture Management Design – An Empirical Analysis. To appear in Proc. Wirtschaftsinformatik 2011. ACM, New York (2011)
33. Hair Jr., J.F., Black, W.C., Babin, B.J., Anderson, R.E., Tatham, R.L.: Multivariate Data Analysis, vol. 6. Pearson Prentice Hall, Upper Saddle River (2006)
34. Aziz, S., Obitz, T., Modi, R., Sarkar, S.: Enterprise Architecture: A Governance Framework - Part I: Embedding Architecture into the Organization, Infosys (2005)
35. Aziz, S., Obitz, T., Modi, R., Sarkar, S.: Enterprise Architecture: A Governance Framework - Part II: Making Enterprise Architecture Work within the Organization. Infosys Technologies Ltd. (2006)
36. Winter, R.: Design of Situational Artefacts – Conceptual Foundations and their Application to IT/Business Alignment. Will appear in Proc. ISD 2010 (2010)

A Method Base for Enterprise Architecture Management

Sabine Buckl, Florian Matthes, and Christian M. Schweda

Technische Universität München, Institute for Informatics,
Boltzmannstr. 3, 85748 Garching, Germany
{sabine.buckl,matthes,christian.m.schweda}@mytum.de
http://wwwmatthes.in.tum.de

Abstract. Responding to the increasing complexity and diversity of information systems development, method engineering provides techniques and tools for the analysis, design, and evolution of information systems. Similar challenges can be found in the research area of enterprise architecture (EA) management, whose main goal is to enhance the alignment of business and IT. While a multitude of methods and models to support EA management have been proposed over the years, situational factors as the goals pursued or the organizational context, in which the management function has to be embedded, are typically neglected.

In this paper, we present a building block base for the design of situated EA management functions based on a comprehensive collection of best practice methods and models for EA management. Therefore, we discuss related work from the area of situational method engineering and pattern-based development and design. Based on these foundations, a building block base and the contained building blocks are presented and are applied alongside a case study from industry. The discussion is complemented by a prototypic tool implementation, which can be used to support the configuration process for situational methods.

1 Motivation

The enterprise forms a complex structure constituted of a large number of highly interdependent elements. The constant need to adapt this structure in response to changing external influences, as economic factors or new regulations, calls for an embracing approach to control and govern the necessary transformations. Enterprise architecture (EA) management is a discipline aiming to provide guidance for the enterprise transformation by taking a holistic perspective on the enterprise, covering concepts from the business to the IT infrastructure level, but also accounting for cross-cutting aspects as strategies, projects, or standards.

The embracingness of the management subject raises different implications relevant to EA management as a function. Most obvious, the holistic perspective taken requires a large amount on information about the architecture elements as well as their interdependencies. Collecting the relevant information, but also keeping the information up-to-date, communicating it to the interested parties

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 34–48, 2011.

(*stakeholders*) in the organization, or performing analyses are tasks, whose complexity grows with the rising amount of information to handle. This and the plurality of possible stakeholders as well as goals to pursue are two reasons that have promoted the development of the plethora of EA management approaches as found in today's literature. These approaches, e.g. The Open Group Architecture Framework (TOGAF) [29], the Archimate language [20], or Core Business Metamodel [22], encompass general prescriptions on how to manage the EA together with conceptual meta-models, the so-called *information models*, for the corresponding management body. Accounting for the fact that each organization has its specific understanding of EA management and the associated goals, the general prescriptions are often complemented with statements that highlight the need to adapt to the using organization. When it nevertheless comes to concrete artifacts describing the *design* of both organization-specific EA management methods and EA description languages, literature becomes more scarce.

Recent publications of Leppänen et al. [21] and of Riege and Aier [24] emphasized the topic of adapting EA management prescriptions to the specifics of the using organization, delineating potential *contingency factors* of EA management. Winter et al. further analyze in [30] the plurality of EA management goals as pursued in practice. These publications indicate towards a more mature understanding of the field (see Section 2 for a detailed discussion). From this dedicated method engineering approaches for EA management can be considered the next research step, which is nevertheless aggravated by the typical challenges of EA management research as outline by Buckl et al. in [8]: practice-relevant EA management research is usually carried out in close cooperation with the industry, such that the projects follow the industry partner's pace and have to deliver their benefits early. On the contrary, the field itself has a broad subject, is rooted in a multi-disciplinary background, and the effects of measures taken usually manifest in the long run after a couple of years. In Section 3 we describe an approach for designing situated EA management functions based on the foundation of method engineering. For the aspect of administering the knowledge base of the approach, we discuss how a pattern-based understanding of EA management, as taken by Buckl et al. in [5], is called upon. Section 4 delineates the steps of applying the approach both from a theoretical perspective and along an anonymized practical example. Final Section 5 summarizes the findings of the article and gives a brief outlook.

2 Related Work

An EA management function can be understood as a design product embedded into the context of using organization. Riege and Aier conducted in [24] an exploratory analysis of the contingency factors, that result from this, and derived a *contingency framework* consisting of three factors:

- *adoption of advanced architectural design paradigms and modeling capabilities*: targeting properties as the coverage of current and target states of the EA in the architectural models as well as of transformation plans

- *deployment and monitoring of EA data sets and services*: concerning the control and governance for the EA management processes via performance reviews and the availability of dedicated EA management marketeers
- *organizational penetration of EA*: aiming at the organizational perception of EA management in the IT departments and business units as well as the usage of EA management-provided services in these departments.

Based on the empiric results on these factors and the constituting items, Riege and Aier cluster the analyzed EA management functions into three different types: *engineering functions*, *incepting functions* and *extended IT architecture functions*. Former distinction supported from the empiric point of view nevertheless reveals a main limitation of the analysis. The analysis' results fail to support a clear distinction between the contingency factors and their effects. This may ascribe to the special nature of the interplay between the EA management function and its management body, but provides only minor support for understanding the impediments and catalysts of managing the EA.

In [21] Leppänen et al. pursue a different approach in deriving the contingency factors of EA management. Based on the findings and experiences of the Finnish EA research program, they elicit their contingency framework (EACon) for EA management, providing the following categories of contingency factors:

- *EA method goals* reflect the stakeholder's requirements that the EA management function is meant to satisfy. These stakeholders can be located in different organizations participating in the EA management.
- *EA principles* delineate constraints pertaining to the EA management function, such governance rules. These principles may be local to one organization or be shared in the organization network.
- *Roles* refer to the people to be involved in both the EA management function as well as in the corresponding governance. Possible roles are the enterprise architect, as *method user*, and the *EA method engineer*.
- *Resources* reflect manpower, monetary supplies and tool support that is available to the EA management function in the participating organizations.
- *Cluster* describes the organizational environment into which the EA management function is to be embedded. Such cluster can be single organization or a network of enterprises cooperating to provide networked services.

Each of the above factors can according to Leppänen et al. [21] be further detailed. For the *cluster* this reads as a more detailed understanding of the *organizational culture* and *organizational structure*, of which the latter is closely related to the factor *decision rights* constituting a part of the *roles*. Making explicit these different factors as well as the intricate relationships inbetween is the core contribution presented in [21] by Leppänen et al. This clearly mirrors the focus of the article, that seeks to contribute to *a contingency framework for engineering an EA planning method*. It is hence not surprising that the particular factors of EACon remain decoupled from concrete solutions, guidelines, or prescriptions on how to optimally manage the EA given certain contingencies.

In [5] Buckl et al. take a different perspective on the field of EA management. Experiencing the need for concrete and practice-proven solutions to recurring

EA management problems, the authors translate the notion of the *pattern* (cf. Alexander et al [2]) to the field of EA management research. For the field of EA management, Buckl et al. introduce three different types of pattern as follows:

- *method pattern* define steps to be taken in order to address a given problem. Furthermore, as a guidance for applying the method, statements about its intended usage context are provided.
- *viewpoint pattern* define the notations used by the methods, i.e. describe ways to present information necessary for performing one or more methods as stored according to one or more information model patterns.
- *information model pattern* supply models structuring the information needed by one or more methods and visualized in one of more viewpoints.

For solving a particular EA management problem in a given organization, a user selects an appropriate set of method, viewpoint, and information model patterns. Based on these constituents, a user composes his specific EA management function, i.e. defines what would in line with Gutzwiller [14] be called an *organization-specific EA management method*. Former term sheds a light on the slightly uncommon understanding of *method* in the work of Buckl et al., e.g. in [5]. While usually design-oriented methods are understood as constituted from *roles*, *tasks*, *techniques*, *design results*, and a corresponding *meta-model*, Buckl et al. separate the roles, tasks and technique (the *method* in their understanding) from the design results with their corresponding meta-model (views according to *viewpoints* and *information models* in their terms). This separation can be justified against the background of the stereotypic tasks employed in EA management and manifests the so-called *method-language-dichotomy* as alluded to e.g. by Schelp and Winter [26] or by Buckl et al. in [8]. This dichotomy is utilized in Section 3 to formulate independent building-blocks for an EA management function in refinement of the EA management patterns.

Helping the user in selecting the suitable EA management pattern, the EA management pattern catalog [11] refines the basic idea and supplies a set of relationships between the different patterns. In particular, every method pattern references all viewpoint patterns that can be used in the method, whereas each viewpoint pattern relates to the information model pattern, covering the needed information. Pattern of all three types are described using a template resembling the so-called *canonical pattern form* (cf. Ernst [12]). Each pattern states:

- its *usage context* in which it can be applied,
- the *problem* that it has proven to solve,
- the *solution* which it applies to the given problem,
- the contradictory *forces* framing the space of observed solutions, and
- the *consequences* observed to result from the pattern's application.

With this standardized structure, the patterns can serve as valuable starting point for developing an approach for designing situated EA management functions. Such approach nevertheless has to deal with the inherent weaknesses of EA management patterns, e.g. their tendency to repeat themselves especially with

respect to methods for documenting the EA, as well as terminological plurality of the pattern descriptions, resulting from the fact that patterns are observed in different practice cases without consistent overarching terminology.

3 Designing a Situated EA Management Function

In line with the understanding of Harmsen that *there is no method that fits all situations* [15, page 6], we subsequently propose a situated approach to design an EA management function based on existing best practices. A situated approach according to Harmsen in [15] accomplishes standardization and at the same time flexibility to match the situation. A *situation* thereby refers to the combination of circumstances at a given point in time in a given organization [15]. In order to address these requirements, for each situation a suitable solution[1] – so-called situational solution – is constructed that accounts for these circumstances. Reflecting the method-language dichotomy in EA management, two different types of solution constituents, *method building blocks* (MBBs) and *language building blocks* (LBBs), are used in the construction process and are configured as well as adapted with the help of formally defined guidelines.

3.1 Foundations

A complex and intricate research area like designing EA management functions represents a topic that is not easy to research. The heavy involvement of stakeholders, the broadness of the subject, and the delayed effects (see Section 1) call for a suitable structuring of the research subject. This structuring can be performed either using a *vertical* or *horizontal* domain decomposition strategy. In the *vertical* domain decomposition only a limited number of EA-related problems is addressed in an embracing manner with a comprehensive solution. In contrast, the *horizontal* domain decomposition addresses a variety of EA-related problems with either suitable management methods or modeling languages.

While the former type of decomposition is a frequently used one for approaching the area of EA management(cf. Johnson and Ekstedt in [17], which focus on EA analysis or Spewak in [28] emphasizing on EA planning aspects), we in line with Schelp and Winter in [26] opt for a horizontal decomposition of the domain of EA management, which reflects the method-language dichotomy as discussed above. Such an approach is further backed by TOGAF, which contains the *architecture development method* – reflecting the methodical perspective – and the content framework – representing the language part (cf. Open Group in [29]). In contrast to the approach taken by TOGAF, we advocate for making the interconnection points between the methodical and the language parts explicit. In order to do so, we introduce the *variable* concept in the MBBs serving as a placeholder for language aspects. *Viewpoint variables* for instance are used during method description to indicate placeholders for visual architecture descriptions, which must be filled during the organization-specific configuration.

[1] We subsequently employ the term solution instead of *method* in line with argumentation at the end of Section 2.

The idea of interrelating best practice fragments to design an organization-specific EA management function can only be realized against the basis of a common understanding and terminology of the topic. Although no such common understanding has yet evolved if the definition of the term *EA* or *EA management* is considered (cf. Schönherr in [27] and Schelp and Winter in [25]), consensus on the fundamental activities and tasks that make up an EA management function exists (cf. [1]). In [10], Buckl et al. revisit different approaches to EA management, e.g. Frank [13], Riege and Aier [24], and The Open Group [29] to devise a method framework for EA management consisting of four activities as shown in Figure 1:

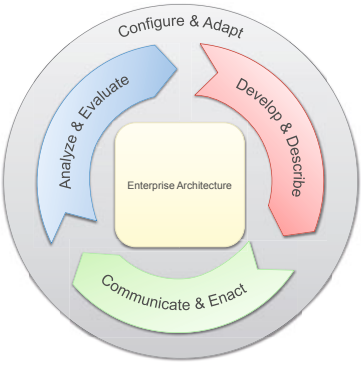


Fig. 1. Method framework of BEAMS

Develop & describe a state of the EA, either a current state describing the as-is architecture, a planned state representing a medium-term future state or a target state, i.e. a vision of the EA.

Communicate & enact architecture states and principles guiding the evolution of the EA to EA-relevant projects and to related management functions, e.g. project portfolio management.

Analyze & evaluate the current state to identify potentials for improvement, evaluate architectural scenarios (planned states), or analyze whether a planned state helps to achieve the target state or not.

Configure & adapt the EA management function itself, e.g. in response to an under achievement of the desired results or a changed situation decide on the addressed management concerns, pursued goals, and used methods.

From the perspective of architectural descriptions, the former activities are characterized as *creating*, *using*, and *augmenting* the EA description. The latter activity – configure & adapt – incorporates the nature of a meta-activity as it is concerned with the design of the former three activities. In terms of the situational method engineering this activity encompasses the *process of situational method engineering* [15, page 45], i.e. the steps *characterization of the situation, selection of method fragments,* and *assembly of method fragments.*

Similar to the method aspect, also the language aspect can be subdivided. An EA management-relevant problem can be described by a concern, i.e. an area of interest, and a goal, i.e. an abstract objective. A concern represents an organization-specific conceptualization of the management subject, while goals complement this static perspective via a time-dependence or a notion of better and worse. In that sense the achievement of a goal, e.g. increase homogenization, can be operationalized via measurements, such that a goal provides an evaluation function, which can be used to guide the EA planning process.

3.2 Structure of the Building Block

As motivated above, two different types of building blocks to design an EA management function exits – MBBs and LBBs, of which the latter are subdivided into building blocks concerned with the conceptualization, i.e. areas of interest – information model building blocks (IBB) – and building blocks containing best practice visual representations – viewpoint building blocks (VBB).

Method building blocks
An MBB describes the different *tasks* that are performed in order to achieve a certain goal in a given organizational *context*. The MBB further specifies the ordering of the tasks and execution alternatives. For every alternative path the MBB also describes the *conditions* that apply during task execution. Furthermore, a task can devise different *techniques* to be utilized. To perform an expert-based analysis of a planned state for example, a pattern-based technique or an-indicator based technique can be utilized. Thus, each technique is linked to *forces* describing the benefits and drawbacks of the different techniques to be selected. Reflecting the method-language dichotomy each MBB contains a *concern variable*, i.e. a placeholder for the area of interest on which the tasks operate. During the configuration, the concern variable has to be replaced by the actual concern.

In order to be applied, the *pre-conditions* specified by an MBB need to be met. An exemplary precondition of an MBB dedicated to the analyze & evaluate activity is that the concern specified by the concern variable is already documented. Supplementary, each MBB also specifies *post-conditions* that are fulfilled after executing the MBB. In this vein, consistency checks can be executed ensuring a sensible configuration and ordering of MBBs. The above exemplary pre-condition illustrates the make-up of pre- and post-conditions, representing a combination of an area-of-interest, i.e. a concern, and a so-called *meat-attribute*, e.g. documented, acknowledged, or publicized, describing a property of the concern related to the MBB. In addition to the post-conditions, an MBB can specify consequences, which result from applying the building block. The notion consequence thereby does not only refer to negative side-effects but is also used to describe positive add-ons. In contrast to the post-conditions, consequences are described in an informal manner utilizing natural language descriptions.

Complementing each MBB contains a *trigger variable*, which specifies the trigger starting the execution of the tasks. In configuring the EA management function this variable is filled with an actual trigger. Each task is executed by a corresponding actor represented by an *actor variable* in the description of the

method. The notion of actor variable similar to the notion of trigger variable is used to denote that the description of the MBB does not specify distinct actors or roles in the using organization, but merely describes a responsibility of a person or group. Further, the MBB can specify that the actor variable is bound in respect to its organizational role, e.g. might express that an escalation based enactment mechanism only works, if a superordinate actor can be called upon. Beside to the mandatory relationship to the executing, i.e. *responsible* actor variable, each task may relate to other actor variables as well, namely variables representing actors that are *consulted* or *informed* during task execution. The distinction between the different levels of involvement pertaining to a single task is based on the RACI model of CobiT (see e.g. [16]), while a slightly different perspective is taken on the involvement level *informed*. For the purpose of describing MBBs, we assume that any actor involved in a task is informed, such that the responsible actor as well as consulted actors are counted as informed, too. The participation of actors in tasks is enabled via *viewpoint variables*, which designate that the actor takes a specific viewpoint on the information relevant during performing the given task. The notion of the variable here again describes that the MBB does not make concrete prescriptions on the viewpoint to be used, but in turn allows to select a specific viewpoint for accomplishing the task.

Language building blocks

In designing the language for EA descriptions, both VBBs and IBBs are employed. Understanding a language in line with e.g. Kühn [18] as constituted of *abstract syntax*, *semantics* and *notation*, the two types of building blocks are used to specify notation and syntax, respectively. The semantics is specified denotationally in the glossary complementing the building block base (cf. Section 3.3 below). Each information model building block specifies the *types*, *attributes* and *relationships* that conceptualize the corresponding part of the EA, i.e. cover a stakeholder's concern or reflect the information necessary for assessing the attainment of a goal. For the latter purpose, the IBBs introduce a distinction between different kinds of types, most notably distinguishing between *classes* and *mixins*. As Buckl et al. outline in [9], latter concepts can be used to formulate specialized IBBs are able to describe specific EA goals as *availability* regardless the actual EA concept, e.g. *business application* or *business capability*, they are attached to. A user can hence select the IBB reflecting a specific concern and combine it with a specific goal to his relevant EA problem, see Figure 2.

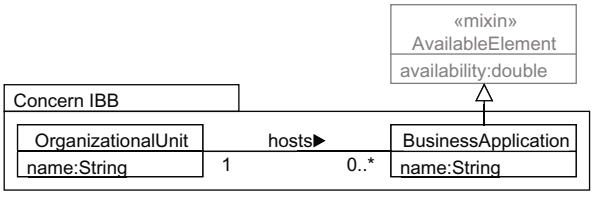


Fig. 2. Integrated information model

The VBBs focus on the notational aspects of the EA description language, providing visual primitives, e.g. *rectangles*, and visualization rules, e.g. *clustering*, for defining how the information is visualized. The VBBs further relate to the information model or parts of it, specifying concepts of which type are mapped to which kind of visualization element. Thereby, an executable transformation from the syntactic concepts to their visual counterparts is defined based on the VBBs. With the focus of this article on the method and information model perspective, we abstain from going into the details of the mechanism behind the transformation. More information can be found in [6].

3.3 Structure and Administration of the Building Block Base

Critical prerequisite to the design of a situated EA management function, is the provision of standardized building blocks, which are stored and retrievable from what is typically called a *method base* (cf. Brinkkemper in [3]) or *component base* (cf. Kumar and Welke in [19]). Due to the method-language dichotomy of our application domain, we abstain from reusing the misleading notions and introduce the term *building block base* for the repository. The structure of this repository is outline below and complemented by a description of the configuration and administration process. Enabling the selection of appropriate building blocks for a given situation requires the development of concepts and techniques to analyze and compare the incorporated building blocks. Following the idea of Pries-Heje and Baskerville in [23], we use the concepts of

problem. A problem represents the issue to be solved by applying the building block. A problem in the area of EA management typically consists of a
  goal representing an abstract objective, e.g. increase homogeneity, provide transparency, and a
  concern, i.e. area of interest in the enterprise, e.g. business support, application systems.

organizational context. The organizational context represents the situation in which the EA management function operates. Typical factors which are considered in the organizational context are the organizational culture, management commitment, or involved stakeholders.

Figure 3 illustrates the components and the structure of the building block base. To outline the administration of the building block base, we exemplify its development along the best practices for EA management as contained in the pattern catalog from [11]. Therefore, the problems addressed by the different patterns are analyzed and the abstract goals as well as the concerns are identified. Thereby, an exemplary goal reads as follows “increase homogeneity” and the respective concern is “technology used by a business application” [4]. The concerns and the respective information model patterns serves as input for the development of IBBs. Thus, also establishing a concern hierarchy, i.e. an evolution path, as introduced in [7]. Furthermore, the usage context descriptions are investigated for descriptions of organizational contexts in which the respective pattern has been applied, e.g. “centralized IT organization”, “upper management support”,

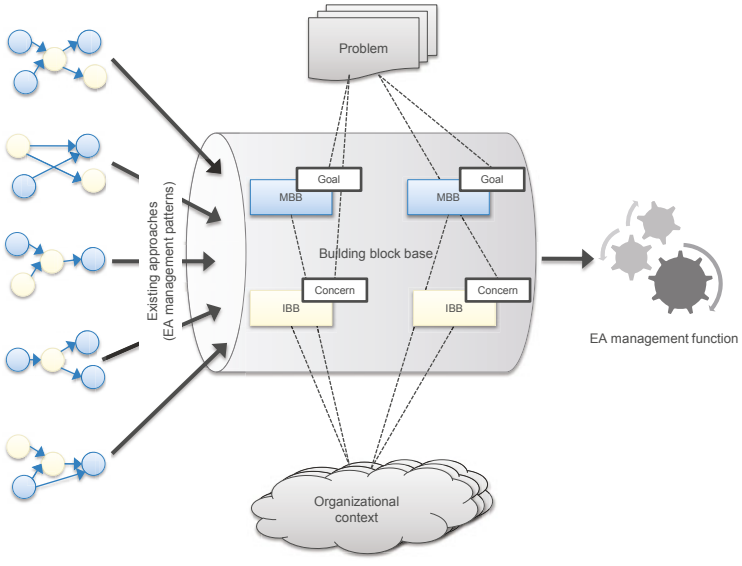


Fig. 3. The components and structure of the building block base

or "own budget for the EA management initiative". The MBBs are derived from existing method patterns, thereby, the textual description of the steps to be taken is used as input to derive tasks, responsible actors, and forces. Furthermore, the consequence section of the patterns serve as input for the pre- and post-conditions of the building blocks. Consequences which can be formalized in terms of a meta-attribute and a respective concern are reformulated as pre- and post-conditions. The identified goals and organizational contexts are used as input for characterizing the situation.

To relate the components of the building block base, the above identified goals of and organizational contexts are the derived building blocks. The suitability of a building block for any combination of the goals and organizational conditions can then be defined utilizing a *fitting matrix* with the building blocks on the y-axis, the identified organizational contexts (goals) on the x-axis, and a scoring of the fitting function for the MBBs (IBBs) in the cell. The fitting function can thereby take a value form the set *required, excludes,* or *helpful* and serves as a decision-support system for the selection of building blocks. The assembly of building blocks is performed utilizing the *variable* concept, i.e. the IBBs are used to configure the concern variable, and the pre- and post-conditions of BBs, which determine an ordering.

4 Applying the Building Block Base

The building block base is applied using three steps: *characterization of the situation*, *selection of building blocks*, and *assembly of building blocks*, described below along an example from a financial service provider BSM.

Characterization of the situation
In this phase the existing organizational context in the enterprise is described. Further, the specific EA management goal to be pursued and the related EA concern are delineated. For doing so, the using organization can call on the list of contexts, goals and concerns as contained in the building block base.

> Resulting from previous acquisitions, BSM operates a highly heterogeneous landscape of business applications. Especially, maintenance of business applications developed a non-standard solutions for the formerly independent companies has become a costly task. Therefore, BSM seeks to *increase homogeneity* of the *business applications hosted at the different locations*. Due to the maintenance problems, the EA management can rely on *high-level management support* and is driven by a small EA management team located in a staff unit of the CIO's office. This team has to deal with the *highly decentralized structure* of the IT departments, such that the EA management process should be design to promote itself.

Selection of fragments
In this phase the building block base is searched for MBBs that match the organizational context and for IBBs that reflect the EA management problem. Figure 4 shows the results of a query against the building block base, returning IBBs containing the concept *application*. Similar searches are performed on the IBBs that reflect the cross-cutting aspect of standardization as well as on the MBBs starting with ones supporting the activity *develop & describe*.

> The search for the applicable methods presents two MBBs as well-suited for the organizational context: *describe by interview* and *describe by workshop*, as both these MBBs are helpful to market the EA management endeavor. For the aspect of standardization, three models for standards reflected in different IBBs are provided by the building block base: *simple standardization*, *standardization via book of standards*, and *standardization by individual prescriptions*. Reading through the consequences of the different building blocks BSM decides to chose an interview-based gathering of information about business applications and their hosting

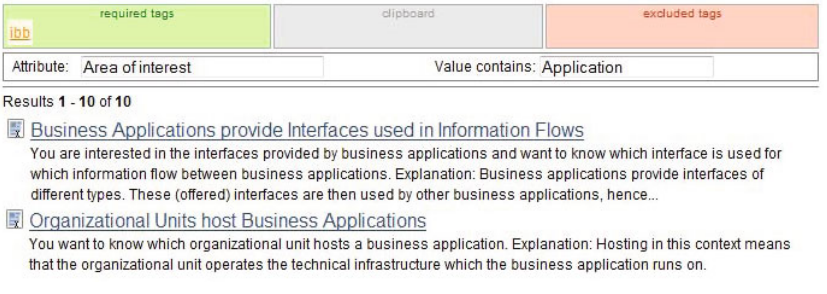


Fig. 4. Searching the building block based for suitable IBBs

organizational units. For the aspect of standardization, a book of standards is to be created, marking certain technologies as standard or non-standard, respectively. For the phase *communicate & enact* the MBB *publish architectural descriptions* is chosen.

Assembly of fragments
In this phase four sub-phases are conducted to compose the building blocks to a comprehensive EA management function:

Integrate IBBs: the different IBBs reflecting the concerns and goals are integrated into composite models covering specific sub-problems of the EA management-problem to address. Thereby, manageable information models are created which can subsequently be linked to the MBBs.

Integrate and configure MBBs: the concern variable of each selected MBB is linked to the integrated information model that reflects the corresponding concern. If different concerns are to be treated by similar methods, the MBBs can be duplicated in this case. The configured MBBs are integrated into a process, and the consistency between the pre-conditions and post-conditions of consecutive MBBs is checked.

Configure actors and triggers: for each sequence of MBBs the triggering event is configured. Further, the actor variables defined by the MBBs are bound to actual organizational roles of the using enterprise.

Add and configure VBBs: for each actor, who participates in a task, a viewpoint variable exists. This variable is set to a composition of VBBs, expressing how certain parts of the corresponding concern (information model) are to be visualized. If the viewpoint variable is designed *read-only*, nearly no limitations on the type of viewpoint exist, whereas a *read-write* viewpoint is bound to comprise VBBs in a combination that represents an *updateable view*. This e.g. means that all information model elements intended to be written are represented 1:1 in the corresponding visualization.

During each of the aforementioned sub-phases, consistency checking is applied. During the integration of the IBBs the approach analyzes, if an incompatible semantic mapping is created, i.e. if *homonyms* in terms of the glossary of the building block base are created by unifying types with a distinct meaning. Concerning the configuration of the VBBs, the approach analyzes whether the visualized information is available in the method's corresponding concern variable. Further, it checks whether the transformation from syntactic to visual primitives is bidirective, or not, such that the latter case is diagnosed as non-updateable view, which cannot be used for write access.

BSM integrates the IBBs covering business applications, hosting organizational units and used technologies into a single information model. In contrast the information, whether a certain technology is standard according to the book of standard, is separated into a different information model. Former model, is assigned to the concern variable of *describe by interview*, whereas latter model is linked to *describe by*

workshop. The post-conditions of the two configure MBBs then read as OrgUnit-BusinessApplication-Technology.*documented* and as Technology.isStandard.*documented*. To the concern variable of the MBB *publish architectural descriptions* the information model subsuming both information models is assigned.

Further concretizing the documentation-specific MBBs, BSM decides that the interviews on the business applications and related information are held every time, when an IT-project finishes. An *enterprise architect* asks the *application owner* of the corresponding business application. The standardization-related information is decided upon every six months during a workshop by a board comprised of *enterprise architects*, *application owners* and *IT project managers*. A re-publishing of the architectural description is triggered every time, when new information about business applications is available. The *enterprise architects* are responsible for creating the corresponding architectural description, which is made available to *application owners*, *IT project managers*, and the *CIO*. With the viewpoint being read-only, BSM decides to use a clustered visualization containing only organizational units and business applications, of which the latter are colored red, if they use at least one non-standard technology, or green otherwise.

5 Summary and Outlook

In this article we motivated the need for organization-specific EA management functions, i.e. management functions that account for the specificities of the using organization with both respect to the context and the management goals. Reflecting on the related work in Section 2, we showed what current research can already contribute to the design of such EA management functions and were able to delineate the omissions of current approaches. In Section 3 we applied the basic notions of method engineering onto the subject of EA management, describing an approach building on three types of building blocks for EA management functions, namely MBBs, VBBs, and IBBs. We further outlined how concrete building blocks look like and how they are interrelated in the *EA management building block base*, a EA management-specific implementation of the notion of the method base. In Section 4 we described how the building block base can be used to design an organization-specific EA management function. Thereby, we employed an example at a financial services provider.

With its roots in the EA management patterns of Buckl et al. [5,11], the approach can rely on a sound and practice-proven basis of best-practice methods, viewpoints and information models. The rigorous mechanism for translating these patterns into building blocks further helps to ensure that the made prescriptions are applicable in practice. Notwithstanding, more in-depth evaluations of the usefulness of the approach remain to be conducted. Thereby, especially a comparison to less formal methods, e.g. TOGAF, are of interest. Such analyses are nevertheless subject for future research. In the context of the necessary

long-term analyses, it would further be interest to analyze, if the approach's prescriptions are beneficially for governing the EA management function, i.e. for evolving the function in response to organizational changes.

References

1. Aier, S., Buckl, S., Franke, U., Gleichauf, B., Johnson, P., Närman, P., Schweda, C.M., Ullberg, J.: A survival analysis of application life spans based on enterprise architecture models. In: 3rd International Workshop on Enterprise Modelling and Information Systems Architectures, Ulm, Germany, pp. 141–154 (2009)
2. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A Pattern Language. Oxford University Press, New York (1977)
3. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information and Software Technology 38(4), 275–280 (1996)
4. Buckl, S., Ernst, A.M., Lankes, J., Matthes, F., Schweda, C.M.: Enterprise architecture management patterns – exemplifying the approach. In: The 12th IEEE International EDOC Conference (EDOC 2008), Munich, Germany. IEEE Computer Society, Los Alamitos (2008)
5. Buckl, S., Ernst, A.M., Lankes, J., Schneider, K., Schweda, C.M.: A pattern based approach for constructing enterprise architecture management information models. In: Wirtschaftsinformatik 2007, pp. 145–162. Universitätsverlag Karlsruhe, Karlsruhe (2007)
6. Buckl, S., Gulden, J., Schweda, C.M.: Supporting ad hoc analyses on enterprise models. In: 4th International Workshop on Enterprise Modelling and Information Systems Architectures (2010)
7. Buckl, S., Matthes, F., Schweda, C.M.: Conceptual models for cross-cutting aspects in enterprise architecture modeling. In: 5th International Workshop on Vocabularies, Ontologies, and Rules for the Enterprise, VORTE 2010 (2010)
8. Buckl, S., Matthes, F., Schweda, C.M.: From ea management patterns towards a prescriptive theory for desinging enterprise-specific ea management functions – outline of a research stream. In: Multikonferenz Wirtschaftsinformatik (MKWI 2010), Göttingen, Germany, pp. 67–78 (2010)
9. Buckl, S., Matthes, F., Schweda, C.M.: A technique for annotating EA information models. In: Barjis, J. (ed.) EOMAS 2010. LNBIP, vol. 63, pp. 113–127. Springer, Heidelberg (2010)
10. Buckl, S., Matthes, F., Schweda, C.M.: Towards a method framework for enterprise architecture management – a literature analysis from a viable system perspective. In: 5th International Workshop on Business/IT Alignment and Interoperability, BUSITAL 2010 (2010)
11. Chair for Informatics 19 (sebis), Technische Universität München. Eam pattern catalog wiki (2010), http://eampc-wiki.systemcartography.info (cited 2010-07-01)
12. Ernst, A.M.: A Pattern-Based Approach to Enterprise Architecture Management. PhD thesis, Technische Universität München, München, Germany (2010)
13. Frank, U.: Multi-perspective enterprise modeling (memo) – conceptual framework and modeling languages. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS 2002), Washington, DC, USA, pp. 1258–1267 (2002)

14. Gutzwiller, T.A.: Das CC RIM-Referenzmodell für den Entwurf von betrieblichen, transaktionsorientierten Informationssystemen. PhD thesis, Universität St.Gallen (1994)
15. Harmsen, A.F.: Situational Method Engineering. PhD thesis, University of Twente, Twente, The Netherlands (1997)
16. IT Governance Institute. Framework Control Objectives Management Guidelines Maturity Models (2009), http://www.isaca.org/Knowledge-Center/cobit (cited 2010-06-18)
17. Johnson, P., Ekstedt, M.: Enterprise Architecture – Models and Analyses for Information Systems Decision Making. Studentlitteratur, Pozkal (2007)
18. Kühn, H.: Methodenintegration im Business Engineering. PhD thesis, Universität Wien (2004)
19. Kumar, K., Welke, R.J.: Methodology engineering: A proposal for situation-specific methodology construction. In: Challenges and Strategies for Research in Systems Development, West Sussex, England, pp. 257–270. John Wiley, Chichester (1992)
20. Lankhorst, M.M.: Enterprise Architecture at Work: Modelling, Communication and Analysis, 2nd edn. Springer, Heidelberg (2009)
21. Leppänen, M., Valtonen, K., Pulkkinen, M.: Towards a contingency framework for engineering and enterprise architecture planning method. In: 30th Information Systems Research Seminar in Scandinavia (IRIS), pp. 1–20 (2007)
22. Österle, H., Winter, R., Hoening, F., Kurpjuweit, S., Osl, P.: Business Engineering: Core-Business-Metamodell. Wisu – Das Wirtschaftsstudium 36(2), 191–194 (2007)
23. Pries-Heje, J., Baskerville, R.: The design theory nexus. MIS Quarterly 32(4), 731–755 (2008)
24. Riege, C., Aier, S.: A contingency approach to enterprise architecture method engineering. In: Feuerlicht, G., Lamersdorf, W. (eds.) ICSOC 2008. LNCS, vol. 5472, pp. 388–399. Springer, Heidelberg (2009)
25. Schelp, J., Winter, R.: On the interplay of design research and behavioral research – a language community perspective. In: Proceedings of the Third International Conference on Design Science Research in Information Systems and Technology (DESRIST 2008), Westin, Buckhead, Atlanta, Georgia, USA, May 7-9, pp. 79–92. Georgia State University, Atlanta (2008)
26. Schelp, J., Winter, R.: Language communities in enterprise architecture research. In: DESRIST 2009: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, pp. 1–10. ACM, New York (2009)
27. Schönherr, M.: Towards a common terminology in the discipline of enterprise architecture. In: Aier, S., Johnson, P., Schelp, J. (eds.) Pre-Proceedings of the 3rd Workshop on Trends in Enterprise Architecture Research, Sydney, Australia, pp. 107–123 (2008)
28. Spewak, S.H., Hill, S.C.: Enterprise Architecture Planning – Developing a Blueprint for Data, Applications, and Technology. John Wiley & Sons, New York (1993)
29. The Open Group. TOGAF "Enterprise Edition" Version 9 (2009), http://www.togaf.org (cited 2010-02-25)
30. Winter, K., Buckl, S., Matthes, F., Schweda, C.M.: Investigating the state-of-the-art in enterprise architecture management method in literature and practice. In: Proceedings of the 5th Mediterranean Conference on Information Systems (2010)

Towards the Use of Granularity Theory for Determining the Size of Atomic Method Fragments for Use in Situational Method Engineering

Brian Henderson-Sellers[1] and Cesar Gonzalez-Perez[2]

[1] School of Software, Faculty of Engineering and Information Technology,
University of Technology, Sydney, P.O. Box 123, Broadway, NSW 2007, Australia
Brian.Henderson-Sellers@uts.edu.au
[2] The Heritage Laboratory (LaPa), Spanish National Research Council (CSIC)
Santiago de Compostela, Spain
cesar.gonzalez-perez@iegps.csic.es

Abstract. Situational method engineering defends the idea that methodologies should be constructed by assembling pre-existing method fragments from a repository. The structure of the repository, the kinds of fragments that it can store, as well as the possible relationships among them, are dictated by an underlying metamodel. One of the aspects that must be studied is that of the granularity of the individual method fragments in the context of the metamodel to which the fragments are conformant. This becomes especially relevant in a service-oriented method engineering context, where interoperability and composability of fragments from multiple repositories is a key issue. This paper applies some theoretical works on granularity to the study of both the granularity and the size of method fragments, recommending some best practices that should be adopted in order that the resultant method fragments are atomic and therefore likely to be consistent in quality thus leading to higher quality constructed methodologies and paving the way for easier composition and interoperation of fragments.

Keywords: granularity, method fragments, situational method engineering.

1 Introduction

Situational method engineering (SME) [1, 2] describes the creation and use of a software development method(ology)[1] from small, atomic methodological pieces, known as "method fragments" or, at a larger scale (i.e. non-atomic), "method chunks" [3], typically conformant to the conceptual definitions in an underpinning metamodel [4].

While much of the literature focusses on method construction e.g. [5-7], little has focussed on the *details* of the atomic elements of a method (method fragments) themselves. In particular, an issue, as yet little discussed, is the *granularity* both of elements in the metamodel and of the atomic modelling elements (method fragments) conformant to it. Firstly, the elements in the metamodel may each have fine

[1] Method and methodology are taken to be synonyms for the purposes of this paper.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 49–63, 2011.

granularity or coarse granularity [8], the latter resulting from an abstraction mapping from a highly detailed system to a less detailed one. This, naturally, affects the granularity of method fragments generated from the metamodel. However, there is a second and orthogonal granularity issue – the granularity and the "size" of the fragment that conforms to such a definition. If fragments are too coarse-grained, thus containing restricted information and/or detail, it is likely that they will highly specific to a single situation (organizational context) i.e. their reusability may be limited and there may be partial overlaps between the specifications of fragment pairs [9]. In addition, it is arguable that fragments coming from repositories constructed on top of metamodels with very different granularities would suffer from interoperability and composability issues, since the abstraction levels at which they have been defined are naturally different. These three granularity issues are highly relevant to issues of SME, e.g. in terms of method construction, fragment storage, fragment interoperability and composability. In particular, a strategic, long-term research goal is an evaluation of the quality of the method fragments and the consequent quality of any methodology constructed from the fragments within the SME approach. In this paper, we concentrate on a precursor for a future quality evaluation by focussing on the granularity of method fragments in the context of their conformance to a metamodel.

Using conclusions from our earlier study of the granularity of metamodels [8], in the context of SME there are two major areas needing to be addressed:

1. the impact on method fragments of the scale and granularity of the metamodel e.g. definitions of method fragment types such as Activity, Task, Step etc. as compared to simply ProcessElement (Figure 1) [similarly Phase, Lifecycle etc.] i.e. the granularity of the metamodel [8].
2. the size of method fragments generated (usually by instantiation) from such method element definitions (made at the meta-level). Here, we focus on fragment generation from the Work Unit metaclass (Figure 2) but note that Work Product and Producer fragments are equally relevant, but are not discussed in any great detail here since analogous arguments apply.

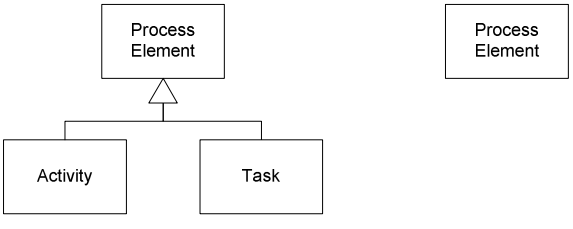


Fig. 1. Granularity of a metamodel might be fine-grained (left) or more coarse-grained (right) (after [8]) With kind permission of Springer Science+Business Media

In this paper, we summarize how to apply a theory of granularity abstraction e.g. [10-13] to the size of method fragments (we focus on fragments as being the only atomic element in SME, eschewing for the present the larger scale method chunk [14] and method component [15]). Following an overview of the theory of granularity (Section 2) and the typical structure of method fragments in Section 3, we then analyze in detail how fragment size and granularity can be optimized using these

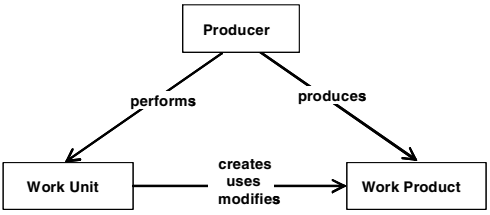


Fig. 2. The triangle of Producer, Work Unit and Work Product that underpins SPEM, OPF and ISO/IEC 24744 standards for software engineering process modelling (after [8]) With kind permission of Springer Science+Business Media

theories. In Section 4, we apply these ideas to a case study of one particular SME methodology: OPEN [16]. We conclude with some recommended guidelines for fragment specification (Section 5).

2 Theory of Granularity

Granularity theory e.g. [10,12] is based on *abstraction* e.g. [11,17,18]. Abstraction, roughly speaking [11], describes an approach in which one focusses on relevant characteristics of a problem (which is often called the "subject under study" (SUS)), whilst discarding its "irrelevant details" [13,18]. These authors [11] propose three informal properties:

1. the abstraction process maps a representation of the problem to a new, more "abstract" representation. (This is the essence of modelling).
2. by throwing away details, the result of the abstraction process provides a simpler problem (a.k.a. "an abstraction") to be solved than the original.
3. by preserving relevant, desirable properties, the solution of the abstracted problem can be transferred to the original, more complex problem, thus solving it.

Since an abstraction, α, necessarily contains less detail than the SUS on which it is based (property 2 above), this may result in several entities in the SUS, e_i, being mapped to a single one in the abstraction i.e.

$$\alpha(e_1) = \alpha(e_2) \neg \Rightarrow e_1 = e_2 \tag{1}$$

[19]. This loss of detail creates a simpler system from the SUS for the purposes of understanding the original SUS e.g. [11,13], although this property is excluded from the formal developments of [11] on account of its complexity. Rather, an abstraction can be defined formally as a mapping, α, between two formal systems, which may be similar or dissimilar e.g. [12]. Here, a formal system, Σ, is a set of formulae, Θ, written in a language Λ i.e.

$$\Sigma = (\Lambda, \Theta) \tag{2}$$

The definition (which addresses property 1 above) given by [11] is:

$$\alpha : \Sigma_1 \Rightarrow \Sigma_2 \tag{3}$$

where the mapping, α, is between a pair of formal systems (Σ_1, Σ_2) with languages Λ_1 and Λ_2 respectively and there is an effective total function, α_Λ, that maps "ground language" Λ_1 to "abstract language" Λ_2 i.e.

$$\alpha_\Lambda : \Lambda_1 \rightarrow \Lambda_2 \tag{4}$$

Of particular relevance to modelling are atomic abstractions, which are fragments that comprise no other method fragments [20] and are therefore instances of the finest granular classes in the corresponding metamodel.

Atomic abstractions can be classified [13] as symbol abstractions, arity abstractions and truth abstractions. Symbol abstractions can operate on constants, functions and predicates and are thus the most relevant to our present discussion. For example, for a symbol abstraction we have

$$x_1, \ldots\ldots x_n \in \Lambda_1, x \in \Lambda_2 \text{ and } \alpha(x_i) = x \text{ for all } i \in [1,n] \tag{5}$$

where x is either a constant, a function or a predicate.

Based on [10], Ghidini and Giunchiglia [13] suggest that symbol abstractions are in fact *granularity abstractions* (e.g. classification, generalization, aggregation). An abstraction α is defined by [12, citing 10 and 11] as a granularity abstraction if and only if

(i) f maps individual constants in Λ to their equivalence class under the indistinguishability relation ~ in Λ.
(ii) f maps everything else, including the predicates in Λ, to itself. (6)

Equation 5 maps *many* individual elements in a set to a single entity in a second set. A natural consequence of this, it is noted, is that granularity abstractions tend to lose information e.g. [21]. As a measure of the degree of granularity, we previously proposed [8] a simple measure: of the system granularity, G_S, as being related to the number of entities, n, in each system. Since it is reasonable to propose that the fine-grained system should have a smaller value for G_S than for a coarse-grained system, we hypothesized that the grain size (system granularity value) is thus a reciprocal measure of the number of granularity abstraction mappings (Equation 5 or 6) between two entities [10]. Thus

$$G_S = 1/n \tag{7}$$

This measure refers to entities represented in a single system or model. As noted above, granularity refers to the degree of decomposition/aggregation, generalization or classification levels often observed in terms of the number and size of extant entities and generally regarded as orthogonal. On the one hand, with a composition granularity abstraction, they take the role of "parts" in a whole-part (aggregation or meronymic) relationship. Thus moving from the "parts" (fine detail) to the "whole" (coarse detail) loses detail, reinforcing the notion that in many senses granularity is a kind of abstraction. In the OO literature, this removal of detail in the process of moving between granularity levels can be modelled not only by a whole-part relationship but also by a generalization relationship between two sets – the generalization granularity abstraction; or by an instance-of relationship between objects and their class – the classification granularity abstraction [21]. Consequently, making such parts or subclasses visible/invisible changes the granularity value of the overall system/model.

Granularity is thus a kind of abstraction that uses aggregation, generalization or classification relationships between entities to achieve simplification. This mechanism produces entities that are more coarse granular than the original, fine grained entities.

Granularity abstractions as applied to metamodels are discussed in [8] where the values of granularity for several current metamodels are given. In the next section, we see how these ideas can be used in the assessment of the granularity and size of method fragments generated to be conformant to a given metamodel.

3 The Granularity of Method Fragments

Situational method engineering relies on the use of stored fragments that each conform to some element in an agreed metamodel, where the metamodel is a model of models e.g. [22] and can thus be thought of as a language [23,24]. While, in principle, a metamodel may focus on the definition of work products, metrics, methodologies etc. (in the software engineering context), here, we assume that the metamodel focuses on software development methodologies, and examine the granularity of those fragments that conform to one particular metamodel element: WorkUnit (Figures 2 and 3).

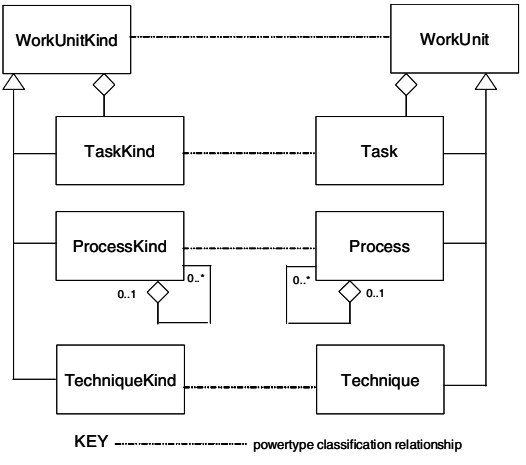


Fig. 3. The WorkUnit/WorkUnitKind metalevel classes together with the subtypes as defined in ISO/IEC 24744

In [8], we examined a number of methodology metamodels in which the values of G_S ranged from 0.25 to 1. Clearly this has a direct impact on the granularity and size of the conformant fragments. Assuming the overall size of the system is S, then if only one fragment were to be generated conformant with each meta-element, then for n meta-elements, the overall size would be related to the fragment sizes by

$$S=\Sigma^{n}_{i=1} f_i \tag{8}$$

where f_i is the size of the i-th fragment. For a constant size, S, this means that the fragment size is bigger for smaller n. Since it is likely that larger fragments are not

atomic and therefore of less than optimal quality, this size evaluation could be contributory to an overall evaluation of the quality of the method fragments in a repository and, by inference, the quality of the constructed methodology.

In this section, we highlight those fragments that depict work unit kinds i.e. what work needs to be done and how – but neglecting the "who", the "when" and the "what" for the sake of simplicity. Each work unit kind fragment is conformant to the WorkUnit/WorkUnitKind metaclass of Figure 2 or one of its subclasses: Process/ProcessKind, Task/TaskKind and Technique/TechniqueKind (Figure 3). We need to note for the discussion in Section 4 that this metamodel permits a Process/ProcessKind to consist of several Task/TaskKind fragments. In the OPEN Process Framework e.g. [6,16], this large agglomeration was known as an "Activity".

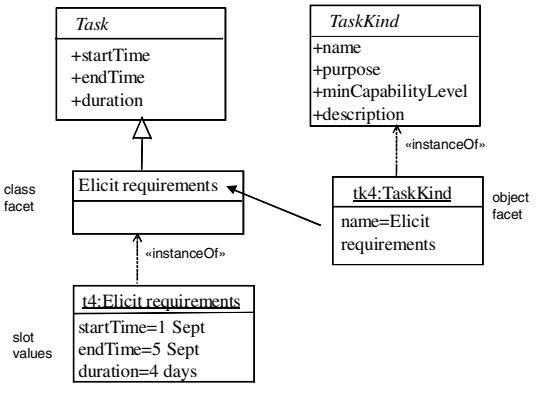


Fig. 4. Powertype pattern for Task/TaskKind and the class and object facets created together with an example of the process "object" in the Endeavour Domain

In the following example, we illustrate the relationship between a fragment and its defining metalevel class with the subclass of WorkUnit/WorkUnitKind named Task/TaskKind. Using ISO/IEC 24744, each concept is depicted in the metamodel using a powertype pattern [25] as shown in Figure 4. Powertype instantiation then depicts, at the model level (i.e. in the Method Domain), an entity with both a class facet (here ElicitRequirements) and an object facet (here t4:TaskKind). The powertype pattern is powerful because it not only permits representations at the model level but also, by instantiating the class facet of the method fragment (in the Model Domain), permits the allocation of project-specific values in the Endeavour Domain [26]. The method fragment is thus a combination of allocated values (from the object facet) and specified but unallocated attributes (from the class facet) (Figure 5).

It is clear from Figures 4 and 5 that any method fragment conformant to the Task/TaskKind powertype pattern that is defined in the ISO/IEC 24744 metamodel will, by definition, contain the exact same number of fields. Whilst most fields will necessarily be brief, the Description field is unconstrained. The long-term research question therefore devolves to an evaluation of "How long (number of words/number of concepts etc.) should a method fragment (such as Task/TaskKind in Figure 5) be in order for the fragment to be regarded as of "good quality"?"

To begin to answer this question, rather than simply a length evaluation, one should consider whether the Description really describes an atomic concept or not. In a particular context, we can determine whether or not a concept is atomic, such that it could be regarded as being of good quality, by "inverting" Equations 5, 7 and 8 to seek (a) a maximum value of n and, in parallel, (b) a minimum value for each f_i. In other words, we seek the set with the largest number of elements that satisfies Equation 5 and a parallel set $\{f_1 \ldots\ldots f_n\}$ such that each f_i is a minimum, whilst retaining a conformance of each of these elements to the relevant class in the metamodel.

Task Kind

Name:	Elicit requirements
Purpose:	To develop and refine a formal and stable requirements specification.
Minimum capability level:	1
Description:	Requirements are to be elicited from clients, domain experts, marketing personnel and users. Usual sub-tasks include defining the problem, evaluating existing systems, establishing user requirements, establishing distribution requirements and establishing database requirements.

startTime
endTime
duration

Fig. 5. The details of a Task/TaskKind fragment called Elicit Requirements

For example, a Task/TaskKind fragment for "Draw a use case diagram" could be considered atomic[2]. In contrast, one could argue that "Create a design for an atomic reactor control system" will necessarily involve a large number of (sub)tasks. Thus, if we are able to break down the fragment into a larger number of other fragments, we can readily deduce that the original fragment was not atomic and hence of poor quality. Adding a quantitative value to that "quality" does not, however, seem possible at this time. This is similar to the discussions well over a decade ago regarding "How big is a good quality object (or rather coded class) when using an object-oriented programming paradigm?" Indeed, our earlier research in this area [27] suggested strongly that there can never be an absolute cutoff threshold number but rather that there is a distribution that can be analyzed statistically such that the larger the size, the lower the probability (but not zero) that the class (and, by extension here, the method fragment) is of good quality.

In the next section, as an exemplar we investigate the current sizes of those fragments stored in the OPF repository [16], specifically those conformant to the WorkUnit/WorkUnitKind meta-element and its subtypes of Process/ProcessKind (OPEN's Activity) and Task/TaskKind (also called Task in OPEN). We analyze these in terms

[2] Although one could argue that it could be broken down as "Draw a symbol for each use case", "Add actors to use case diagram" etc., nevertheless in terms of the atomicity of "Task" it is reasonable to take as atomic since these more detailed elements such as "Add actors to use case diagram" are either Steps within the Task or associated Techniques linked with the Task.

of the granularity theory outlined in this and previous sections before making recommendations to improve the overall consistency in terms of their granularity.

4 Case Study Based on the OPF Metamodel and Fragment Definitions

4.1 The Current Situation

In the original published version of OPEN [16], modelling was seen as beginning to subsume and replace the subactivities of object-oriented analysis and object-oriented design. At the time, the general ideas of using models were gaining strength as a result of the publication of the Object Management Group's Unified Modeling Language or UML [28]. Despite the use of the word "modelling" within the subactivity called Evolutionary Development, itself embedded within the Build Activity of the OPEN Process Framework (as it was later renamed in [6]), in the formal description, modelling was really captured totally in the Task: *Construct the object model* (described in more detail in [16, p 160-162]) – see abbreviated version in the Appendix). However, the description of this "task" was extensive in both detail and scope and, indeed, in a later publication [6], the increasing size of this task was noted (page 274) where it is stated: "In this fairly Large-scale Task, …". Indeed, the description of this particular task includes a suggested list of supportive techniques that can be used to accomplish this task. The number of suggested techniques is 37. These are clearly not 37 alternatives from which one is to be chosen but rather a suite from which strictly more than one is necessary. Thus, this task cannot be considered to be atomic – as argued above, atomicity should be the goal. Thus, for instance when the model is of a Task implemented by a Technique (as in the OPF metamodel – now replaced by ISO/IEC 24744 [29]) – then there should probably (or at least on most occasions) be only one technique for each task – or at worst a *choice* of one technique from a possible suite of alternatives. What is not correct is that there should be a concatenation or suite of techniques that are mandatory to accomplish the task.

The Task *Construct the object model* is described [16] as "the prime technical focus of any OO methodology" where a number of model-descriptive diagrams are constructed. Six diagrams were listed (pre-UML) and of course now the whole gamut of 13 UML 2 diagram types would require support. The fact that so many diagram types are involved suggests strongly that the granularity of this current Task is much too coarse and that its scope is more akin to that of an OPF Activity than a single Task.

These granularity problems were compounded when the OPF repository of method fragments was extended to support agent-oriented method construction. During an extensive analysis of a large number of existing agent-oriented methodologies (summarized in [30]), a new OPF Task: *Construct the agent model* was introduced in parallel to the existing Task: Construct the object model. As more agent-oriented (AO) methods were analyzed, this AO Task grew in size. Thus, for instance, in the analysis of the PASSI methodology e.g. [31-33], which followed eight other analyses, a mapping from several PASSI tasks to the OPF Task: *Construct the agent model* was made. Specifically, Henderson-Sellers *et al.* [33] identified (from PASSI):

- Agent identification
- Task specification (actually suggested as a subtask)

- Ontology description (also OPF Task: Define ontologies)
- Role description (needing some extension)
- Agent structure definition (plus additional subtask)
- Agents behaviour description (new subtask)

That six PASSI tasks were covered by a single OPF task immediately suggests a granularity problem with the OPF Task: Construct the agent model – since, by definition, a task is atomic, being the smallest fragment that can be project managed.

The current situation is thus that there are two OPF Tasks: *Construct the object model* and *Construct the agent model* (Figure 6(a)). These two tasks would appear at first glance to have the same focus and scope but to be applied to two different technologies (the object model and the agent model). It is therefore appropriate to challenge the need for two such tasks as an exemplar of some consequences of imprecise definition of appropriate "granularity". Consequently, we seek to evaluate the efficacy of uniting these two originally disparate tasks. We show that, by focussing on *modelling*, we can ensure that a method fragment is created that is technology independent. Of course adding to the existing *Construct the object model* will make it even larger (Figure 6(b)), thus adding to the problems identified in the previous paragraph. In other words, using the theoretical discussions in earlier sections of this paper, we investigate (in Section 4.2 below) whether the converged modelling task outlined above is at the appropriate granularity.

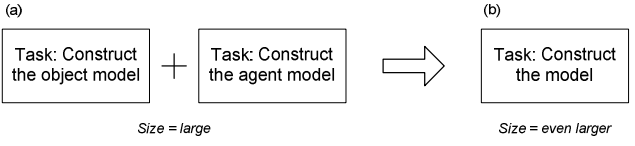


Fig. 6. Amalgamating two existing tasks, both already too large, creates an unacceptably large "tasks". Part (a) reflects the left hand side of Equation 5 with n=2 whilst part (b) represents the single element (x) with n=1.

4.2 The Proposed New Situation

In this section, we investigate two issues: (i) replacing the "bloated" *Construct the object/agent model* tasks of the original OPF with a larger set of much smaller granularity (n much larger or G much smaller: (Equation 7)) by ensuring that each task is an atomic abstraction i.e. a granularity abstraction (Equations 5 and 6) and (ii) seeking a balance between tasks and activities in this modelling area. These two issues are considered in the two following subsections.

4.2.1 New Tasks Derived from the Two Existing Modelling Tasks

We now examine the text in Appendix, which has been abstracted from the pertinent parts of the original description in [16]) for the Task: *Construct the object model*. We undertake essentially a textual analysis by identifying nouns that represent work products that need a task to produce them and verbs that represent the actions that are the core of a task. The list we have constructed includes:

- Identify classes and objects
- Identify roles to be played by objects
- Identify responsibilities of each class
- Add stereotypes[3]
- Implement responsibilities as class operations/methods and attributes
- Identify class-class relationships including possible meronymic (whole-part) representations
- Identify inheritance hierarchies
- Add constraints such as cardinalities
- Specify object behaviour, including its lifecycle
- Define state transition diagrams for each class, as necessary.

Then, from our studies of agent modelling, we can identify tasks such as

- Identify each agent in the system-to-be
- Specify the tasks associated with each agent
- Describe the roles that an agent may play in the system-to-be
- Describe the ontology associated with each agent
- Define the internals of each agent (agent structure)
- Design the behavioural aspects of each agent
- Design the interactions between agents

These two lists are of course peers but provide selection lists affiliated with the decision on which technology, objects or agents (or possibly a hybrid architecture) is to be used in any particular situation i.e. in constructing a particular situational method. The tasks are all atomic although space precludes formal proof of this.

4.2.2 Elevation to Activity Status and Merger with a Pre-existing Activity

Although tasks are the main focus of the work unit idea in the OPF, there is also an element called Activity. The need for an activity is twofold: firstly, to gather together a number of tasks with a "placeholder" that is at a higher abstraction level and, secondly, to be used in full lifecycle method construction on the grounds that, whilst a full-scale method may need many tens if not hundreds of tasks (and associated techniques), it will only have a handful (around a dozen or so) elements of the granularity of an Activity.

We therefore argue for the introduction of a new concept at a higher abstraction level than the tasks described above (i.e. an activity) called *Construct the model using the selected technology/paradigm*. Such a "promotion" would be in line with the philosophy underpinning the MDA and model transformations as well as the use of meronymy for creating a granularity abstraction hierarchy. This new activity then consists of a large number of tasks, these tasks being those listed above in Section 4.2.1 where each one meets the notion of abstraction atomicity defined in Section 3.

There is, however, one further technical discussion point in that there is a pre-existing Activity in the OPF called "Design" [6] that has these modelling concerns as one of its tasks. By elevating the model construction to an activity granularity, we in-

[3] Only when necessary and appropriate (ensuring correct definitions of each stereotype are available).

troduce a conflict of terminology. However, since publication, the OPF metamodel has been replaced by that of ISO/IEC 24744 and, although we have retained the terminology of the original OPF by using the name Activity rather than the name Process as preferred in ISO/IEC 24744, we can now take advantage of the recursive relationship in ISO/IEC 24744 that supports both processes and sub-processes (Figure 3) and thus consider the newly introduced *Construct the model using the selected technology/paradigm* as a subactivity or subprocess of the OPF *Design Activity/Process*.

5 Discussion and Related Work

Abstraction has long been recognized as a keystone of software engineering modelling [34, 35 ch. 3]. Two fundamental abstraction mechanisms are stated in [36] as being composition and classification. Whilst both are useful in discussing granularity [21], we have here concentrated on the former mechanism in our example of a Process/ProcessKind fragment in terms of an aggregation of Task/TaskKind fragments.

The granularity of method fragments in SME is discussed in [37, 38] and used by [39] – five categories are proposed qualitatively: method, stage, model, diagram and concept. Their finest granularity level (concept) is akin to the notion of atomicity discussed here. In the (object-oriented) methodological research literature, to the best of our knowledge, no-one has attempted to underpin discussions of granularity with theory – as discussed in Section 2 here. Indeed, Bettini and Ruffini [40] note, as do we, a history dating back to around 1985 with the publication of Hobbs [10] – although the focus in [40] is on the introduction of temporal constraints into discussions of granularity (and therefore out of scope for our discussion in this paper). In Unhelkar and Henderson-Sellers' [41] discussion of granularity, they focus on a qualitative evaluation of object-oriented designs in the context of reuse.

The introduction of two levels of granularity as in Section 4.2.2 (which distinguishes between the coarse granular entity labelled Activity and the finer granularity of the Tasks and their meronymic relationship) is a direct reflection of the basic granularity abstraction as defined in Equation 5. In practical terms it allows the use of both Tasks (small granularity, G) and elements, called here Activities (or Processes if using ISO/IEC 24744), of a larger granularity. However, as seen from Equations 5, 7 and 8, there is no unique mapping between these granularities. In the context of the application of granularity theory as expressed by these equations to SME situations such as described in Section 4.2.2, there remains an ambiguity regarding whether a given task or collection of Tasks is or is not at a high enough granularity (large value of G) to be called an Activity. This is seen for instance, qualitatively, in the discussions in [6] in their discussion of Environment Engineering and Project Management.

6 Conclusions and Recommendations

In this paper we have analyzed the theory of granularity in the context of method fragments used in situational method engineering and argued that it is important that each method fragment has a granularity that is atomic i.e. it cannot be broken down into a meronymically-based hierarchy i.e. the value of G (Equation 7) has been minimized. We have also argued that consistently-sized fragments should enhance composability

and interoperability of fragments across repositories in a service-oriented method engineering context, although this is a topic for further, future elucidation.

We have applied these ideas to one specific set of method fragments – those found in the repository of the OPEN Process Framework and documented in several books and research papers. We have taken as an illustration a single example – the previously documented concern that the task called *Construct the Object Model* supplemented by the task *Construct the Agent Model* had accreted to the extent that it can no longer be regarded as atomic. We have therefore analyzed the documented descriptions of these two so-called tasks and identified a larger number of smaller granularity tasks (atomic granularity) and propose that these should replace the two earlier tasks in the OPF repository.

Future work entails applying the same idea to all the other tasks within the OPF repository to locate any other tasks that are no longer atomic. This size increase is likely to have occurred when new software ideas, such as the introduction of web technologies and aspects, were included by the expediency of adding information to pre-existing method fragments, rather than the introduction of brand new (atomic size) method fragments. We therefore recommend for future work the systematic review of all method fragments in the OPF repository and, similarly, for other pre-existing method fragment repositories from other authors. Determining size and atomicity properties across repositories, even as aggregated values, is also suggested as a tentative way to document the abstraction level at which the concepts that underpin each particular repository have been captured. This should be explored from the perspective of tool-assisted composition of method fragments in a service-oriented SME context.

Of more widespread applicability are questions that could form the topic for future research that relates to the effect of changing granularity on the usability of methodologies. In addition, a more detailed study of the effects of granularity on reusability of fragments in comparison to chunks, for example building on the studies in [9], would make a valuable contribution to the SME literature.

Acknowledgements

This is paper number 10/06 of the Centre for Object Technology Applications and Research within the Centre for Human Centred Technology Design of the University of Technology, Sydney.

References

1. Welke, R., Kumar, K.: Method Engineering: A Proposal for Situation-Specific Methodology Construction. In: Cotterman, W.W., Senn, J.A. (eds.) Systems Analysis and Design: A Research Agenda. J. Wiley & Sons, Chichester (1991)
2. Brinkkemper, S.: Method Engineering: Engineering of Information Systems Development Methods and Tools. Inf. Software Technol. 38(4), 275–280 (1996)
3. Ågerfalk, P.J., Brinkkemper, S., Gonzalez-Perez, C., Henderson-Sellers, B., Karlsson, F., Kelly, S., Ralyté, J.: Modularization Constructs in Method Engineering: Towards Common Ground? In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences. Proceedings of the IFIP WG 8.1 Working Conference. IFIP, vol. 244, pp. 359–368. Springer, Boston (2007)

4. Henderson-Sellers, B.: Method Engineering for OO System Development. Comm. ACM 46(10), 73–78 (2003)
5. Ralyté, J., Rolland, C.: An Assembly Process Model for Method Engineering. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 267–283. Springer, Heidelberg (2001)
6. Firesmith, D.G., Henderson-Sellers, B.: The OPEN Process Framework. An Introduction. Addison-Wesley, Harlow (2002)
7. Henderson-Sellers, B.: Process Metamodelling and Process Construction: Examples using the OPEN Process Framework (OPF). Annals of Software Engineering 14, 341–362 (2002)
8. Henderson-Sellers, B., Gonzalez-Perez, C.: Granularity in Conceptual Modelling: Application to Metamodels. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 275–288. Springer, Heidelberg (2010)
9. Henderson-Sellers, B., Gonzalez-Perez, C., Ralyté, J.: Comparison of Method Chunks and Method Fragments for Situational Method Engineering. In: Procs. 19th Australian Software Engineering Conference. ASWEC 2008, pp. 479–488. IEEE Computer Society, Los Alamitos (2008)
10. Hobbs, J.: Granularity. In: Procs. Int. Joint Conf. on Artificial Intelligence, IJCAI 1985 (1985)
11. Giunchiglia, F., Walsh, T.: A Theory of Abstraction. Artificial Intelligence 57(2-3), 323–390 (1992)
12. Mani, I.: A Theory of Granularity and its Application to Problems of Polysemy and Underspecification of Meaning. In: Cohn, A.G., Schubert, L.K., Shapiro, S.C. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR 1998), pp. 245–257. Morgan Kaufmann, San Mateo (1998)
13. Ghidini, C., Giunchiglia, F.: A Semantics for Abstraction. In: López de Mántaras, R., Saitta, L. (eds.) Procs. 16th European Conf. on Artificial Intelligence, ECAI 2004, pp. 343–347. IOS Press, Amsterdam (2004)
14. Ralyté, J., Rolland, C.: An Approach for Method Engineering. In: Kunii, H.S., Jajodia, S., Sølvberg, A. (eds.) ER 2001. LNCS, vol. 2224, pp. 471–484. Springer, Heidelberg (2001)
15. Wistrand, K., Karlsson, F.: Method Components – Rationale Revealed. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 189–201. Springer, Heidelberg (2004)
16. Graham, I., Henderson-Sellers, B., Younessi, H.: The OPEN Process Specification. Addison-Wesley, Harlow (1997)
17. Kaschek, R.: A Lttle Teory of Astraction. In: Rumpe, B., Hesse, W. (eds.) Modellierung 2004. Proceedings zur Tagung in Marburg/L. LNI, vol. 45, pp. 75–92. Springer, Berlin (2004)
18. Keet, M.: Enhancing Comprehension of Ontologies and Conceptual Models through Abstractions. In: Basili, R., Pazienza, M.T. (eds.) AI*IA 2007. LNCS (LNAI), vol. 4733, pp. 813–821. Springer, Heidelberg (2007)
19. Kühne, T.: Matters of (Meta-)modeling. Softw. Syst. Model. 5, 369–385 (2006)
20. Cervenka, R., Trencansky, I.: AML. The Agent Modeling Language. Birkhäuser, Basel (2007)
21. Keet, C.M.: A Taxonomy of Types of Granularity. In: Procs. IEEE International Conference on Granular Computing (GrC 2006), Atlanta, USA, May 10-12, pp. 106–111. IEEE Computer Society, Los Alamitos (2006)
22. Favre, J.-M.: Foundations of Model (Driven) (Reverse) Engineering: Models. Episode I: Stories of The Fidus Papyrus and of The Solarus. In: Bézivin, J., Hockel, R. (eds.) Procs. Dagstuhl Seminar 04101 “Language Engineering for Model-Driven Software Development” (2005)

23. Gonzalez-Perez, C., Henderson-Sellers, B.: Metamodelling for Software Engineering. J. Wiley & Sons, Chichester (2008)
24. Bertoa, M.F., Vallecillo, A.L.: Quality Attributes for Ssoftware Metamodels. In: Procs 13th TOOLS Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2010). IEEE Computer Society Press, Los Alamitos (2010) (in Press)
25. Henderson-Sellers, B., Gonzalez-Perez, C.: Connecting Powertypes and Stereotypes. J. Object Technol. 4(7), 83–96 (2005)
26. Gonzalez-Perez, C., Henderson-Sellers, B.: A Powertype-based Metamodelling Framework. Software and Systems Modeling 5(1), 72–90 (2006)
27. Haynes, P., Henderson-Sellers, B.: Cost Estimation of OO Projects: Empirical Observations, Practical Applications. American Programmer 9(7), 35–41 (1996)
28. OMG: UML Semantics, Version 1.0, OMG document ad/97-01-03 (January 13, 1997)
29. ISO/IEC: Software Engineering – Metamodel for Software Development. ISO/IEC 24744, Geneva, Switzerland (2007)
30. Henderson-Sellers, B.: Creating a Comprehensive Agent-oriented Methodology - Using Method Engineering and the OPEN Metamodel. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, ch. 13, pp. 368–397. Idea Group, Hershey (2005)
31. Burrafato, P., Cossentino, M.: Designing a Multi-agentSsolution for a Bookstore with the PASSI Methodology. In: Procs. 4th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS 2002), Toronto (May 2002)
32. Cossentino, M.: From Requirements to Code with the PASSI Methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, pp. 79–106. Idea Group, Hershey (2005)
33. Henderson-Sellers, B., Debenham, J., Tran, N., Cossentino, M., Low, G.: Identification of Reusable Method Fragments from the PASSI Agent-oriented Methodology. In: Kolp, M., Bresciani, P., Henderson-Sellers, B., Winikoff, M. (eds.) AOIS 2005. LNCS (LNAI), vol. 3529, pp. 95–110. Springer, Heidelberg (2006)
34. Hazzan, O., Kramer, J.: Abstraction in Computer Science and Software Engineering: A Pedagogical Perspective (2006), http://edu.technion.ac.il/Faculty/OritH/HomePage/FrontierColumns/OritHazzan_SystemDesigFrontier_Column5.pdf (accessed 28.4.2010)
35. Meyer, B.: Object-Oriented Software Construction, 2nd edn. Prentice-Hall, Englewood Cliffs (1997)
36. Jørgensen, K.A.: Modelling on Multiple Abstraction Levels. In: Procs. 7th Workshop on Product Structuring – Product Platform Development, Chalmers University of Technology, Göteborg (2004)
37. Brinkkemper, S., Saeki, M., Harmsen, F.: Assembly Techniques for Method Engineering. In: Pernici, B., Thanos, C. (eds.) CAiSE 1998. LNCS, vol. 1413, pp. 381–400. Springer, Heidelberg (1998)
38. Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-Modelling Based Assembly Techniques for Situational Method Engineering. Information Systems 24(3), 209–228 (1999)
39. Sunyaev, A., Hansen, M., Kremar, H.: Method Engineering: A Formal Description. In: Papadopoulos, G.A., Wojtkowski, W., Wojtkowski, G., Wrycza, S., Zupančić, J. (eds.) Information Systems Development, pp. 645–654. Springer, New York (1999)
40. Bettini, C., Ruffini, S.: Deriving Abstract Views of Multi-granularity Temporal Constraint Networks. In: Hameurlain, A., Cicchetti, R., Traunmüller, R. (eds.) DEXA 2002. LNCS, vol. 2453, pp. 295–317. Springer, Heidelberg (2002)
41. Unhelkar, B., Henderson-Sellers, B.: ODBMS Considerations in the Granularity of a Reusable OO Design. In: Mingins, C., Meyer, B. (eds.) TOOLS 15, pp. 229–234. Prentice Hall, Upper Saddle River (1995)

Appendix: Task: Construct the Object Model

Textual description abstracted from the original source [16].

Associated Techniques
Abstract classes, association, blackboarding, BNF, collaborations, composition structures, connascance, contract specification, data-flow modelling, delegation, ER modelling, event charts, event modelling, formal methods, fuzzification, generalization, genericity specification, hypergenericity, implementation inheritance, information engineering, object lifecycle histories, ownership modelling, partitions, pattern recognition, Petri nets, power types, polymorphism, responsibilities, role modelling, service identification, state machines, stereotypes, task cards, transformations of the object model, usage, use cases, visibility.

Description
…….

Building this single object model can use a wide range of techniques (as listed above) dependent on whether the focus is on task modelling, business object modelling or system object modelling. At each stage, candidate CIRTs[4] are identified and the relationships between them expressed in terms of CIRT responsibilities leading to the use of associations, aggregations, containments, dependencies, collaborations and, later, inheritance structures. It is unnecessary that these relationships be fully defined or be accurate in the cardinality. It is better to draw informal connections (unlabelled and with deferred cardinality) than none at all between CIRTs[5]—mandatory constraints and cardinalities should not be enforced too soon as these are likely to change as the object model is continually refined. We find that often these rough sketches will aid in a rapid elimination of redundant or duplicate CIRTs. Once the initial relationships are identified they should be depicted in the appropriate object-oriented diagrams and fully documented.

…….

Whilst this is more realistically a representational issue, most methodologies, including OPEN, offer a suite of complexity management tools within the guidelines of the method itself. These are aimed at creating a self-consistent suite of diagrams which, together, document the totality of the one model. In earlier methods, the way these various model 'views' linked together was somewhat suspect. It is important that these orthogonal views, often at different abstraction levels (more or less detailed), *all* represent the same 'truth' in the model being created. For example, changing a message send in the dynamic model should change it similarly in its service representation within the CIRT interface; changing the name of a CIRT should be reflected in the use cases and vice versa. This is important particularly when CASE tool support is sought. The tool needs to have a global view despite the fact that any one diagrammatic representation…only shows a subset of the total information available for the model.

[4] In the original OPEN, CIRT was used as supertype of class, instance (object), role and type.
[5] This is possible using the TBD relationship icon in COMN but not possible in UML.

A Method Assessment Framework

Tom McBride and Brian Henderson-Sellers

School of Software, Faculty of Engineering and Information Technology,
University of Technology, Sydney, P.O. Box 123, Broadway, NSW 2007, Australia
{Tom.McBride,Brian.Henderson-Sellers}@uts.edu.au

Abstract. Situational method engineering is used to create methods for use on projects. It is vital that such constructed methods be of good quality and relevant to the software development project in hand. Current capability assessment approaches cannot readily be applied to such SME-constructed methods since they do not differentiate between the three "phases" of method construction and enactment: method design, method enactment and method performance. Here, we clearly differentiate the kind of quality assessment activities that need to be performed in these three different situations.

Keywords: constructed methods, method design, method enactment, method performance, quality assessment, situational method engineering.

1 Introduction

Situational method engineering (SME) [1-3] is the software engineering discipline that describes the creation and use of a software development method(ology)[1] from small, atomic methodological pieces, known as "method fragments" or, at a larger scale (i.e. non-atomic), "method chunks" [4]. Each of these atomic pieces is typically conformant to the conceptual definitions in an underpinning metamodel [5, 6] and stored in a repository or methodbase (Figure 1). Fragments are selected from the repository in order that the final, constructed method (or "process model": Figure 2) will meet the many project contingencies [7, 8] in a way that an off-the-shelf method is not able to do. The constraints on fragment selection are at the heart of situational method engineering and are discussed by many authors (see, e.g. [9-11]). However, there has been almost no discussion of how one might assess the "quality" of the constructed method and little on the quality of individual fragments. A growing number of methods are currently being developed by various communities of interest to support without the aid of method engineers or method engineering[2]. We believe that such initiatives can be supported by a framework that separates the concerns of different phases in the life cycle of a method, identifying what can reasonably be achieved at each phase. Such a framework can assist non-specialists to review their intended method as a quality check and to draw attention to different situational method

[1] Method and methodology are taken to be synonyms for the purposes of this paper.
[2] Many methods are published process reference models in ISO standards. E.g. see ISO 20000-4, ISO 12207:2008, ISO 15288:2008.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 64–76, 2011.

engineering activities that might be necessary. The overall concern of the framework is to address the question of whether or not the method is suitable for its intended purpose in the circumstances for which it is intended.

Method quality could be defined as (a) whether the overall method is complete e.g. whether there are missing work products, work products but no way of creating them and, secondly, as (b) whether the method is actually what is needed for the particular industry application. This latter quality assessment has several aspects, one of the topics discussed in this paper. Firstly, it is possible to assess the quality of the "designed method" although many additional constraints appear when the process model (method) is enacted for a real project (Figure 2). Enactment here is taken to mean the instantiation of the process model for a particular situation or context. This will mean allocating team member names, budgets, time constraints etc. to the "slot values" of the various methodological elements. Finally, an additional quality concern relates to how well the enacted method performs in real time (the "performed process" in Fig. 2). Each of these three "phases" (method design, method enactment and method performance) are the focus of the discussion here.

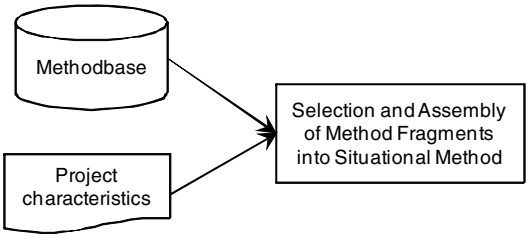


Fig. 1. Engineering a situational method from the elements in the methodbase taking into account the prevailing situation including project characteristics, overall context and other contingencies (after [2])

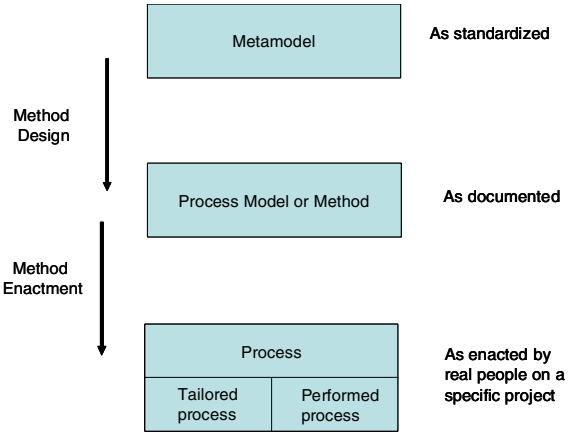


Fig. 2. The "layers" of process and method terminology revised by adding temporality (modified from [12])

This paper will argue that the quality of a method can and should be assessed at the three phases in the life cycle of a method identified above: when the method is designed, when the method is enacted and when the method is being performed. We focus on the creation of a framework to identify "what" needs to be assessed and "when"; but do not attempt to answer the question regarding "how" this assessment will be undertaken in terms of what software engineering metrics might be useful in each phase.

Method design (Section 2) is intended to designate the time when a new method is created, in order to achieve a specific purpose but in a limited range of circumstances. For example, a method engineer might design a new method to develop life critical software using medium to large teams under tight schedule and budget constraints. Method design also includes the case of an organization adopting and possibly adapting a "standard method" for use on all of their projects. Method enactment (Section 3) designates when a method is tailored to the specific circumstances of a project. At that time, decisions can be made to address the project constraints and contingencies. Method performance (Section 4) designates the period over which the method is being used (calendar time) in order to carry out the project and actual performance can be evaluated. In Section 5, we outline the several advantages of this framework over traditional process assessment approaches, such as ISO/IEC 15504 [13] and CMMI [14]. We then, in Section 6, comment on other work (although limited) in this area of method assessment and then conclude in Section 7.

2 Method Design

The focus of this part of the assessment is on the quality of the method fragments/chunks as well as on how well they fit together in the designing of the method.

Software development methods are created to address a range of circumstances. Waterfall development [15], spiral development [16], agile [17] and its related methods of extreme programming [18], adaptive [19] and Scrum [20] are some that have been proposed to deal with particular circumstances or to achieve particular goals. These and others are not intended to be used without modification (although often perceived as being applicable "off-the-shelf") but are offered as a general method suitable as a base from which to tailor a project-specific method [21]. Even though these methods are necessarily general, their usefulness and ability to achieve their primary purpose of developing software in the claimed range of circumstances needs to be assessed. To date, such assessment has been through peer review and commercial acceptance. But just how well any particular software development method is suited to a particular range of circumstances remains very much a matter of personal judgment of the assessor. This introduces a degree of subjectivity into the assessment (see further discussion below).

The earliest attempt to document a method is usually taken to be that of Royce [15] who set out his conclusions from his experiences in developing large software systems. Royce did not describe a software development method in any detail, simply the steps, their interaction, the problems arising within the steps and some ways of overcoming them. Royce addressed concerns arising from managing large software

development projects in general. He did not address specific contingencies such as uncertainty or criticality of the system. JAD and RAD were two of the early attempts to reduce the amount of uncertainty through some initial development to test possible solutions (see e.g. [22] p473). Boehm [16] proposed the spiral method specifically to address project risk while retaining the general collection of software development activities and their inter-relationships. Since Boehm, different approaches have been proposed to address the problem of uncertainty of problem understanding and uncertainty of solution construction. Several software development methods that made use of, and depended on, experienced developers became labelled as "agile" in reference to their claims of being able to respond quickly to the growing understanding of the problem and its implemented solution as all stakeholders learned more about both. Indeed, it is well known (but poorly documented or researched) that organizations adopt and refine their chosen method in some way without making the method specific to a particular project. Assessment of these, and other, software development methods has been largely through commercial acceptance and not through any theoretical or framework based assessment.

In general, some assessment of the method is indeed possible in order to determine if it can address its known or implied range of contingencies. While there are limits to the assessment, it is possible to determine whether or not the method has the means to set and manage performance, to coordinate the work and to develop, deploy and possibly maintain the software, system or service.

The primary purpose of a method is to achieve some purpose, in this case to develop and, possibly, deploy and maintain software, a system or a service. Thus, for assessment purposes the method needs to contain the processes or activities that are capable of achieving the primary purpose. The activities to achieve the primary purpose seem to have been the focus of many proposed situational method engineering methods [1, 23-25].

Processes can, of course, be examined to determine if they contain the expected activities [26]. However, a method is not a random collection of processes nor does it consist only of the processes that support the primary activities. The interaction between processes required to support project management or other forms of coordination can also be examined, as can the process interactions necessary for performance management. For the purposes of this paper, performance management will include those activities involved in setting performance expectations and constraints such as delivery dates, budget and project scope, then managing those expectations and constraints throughout the life of the project.

The software development work must be coordinated, so the method must contain the means to achieve or impose coordination. Coordination is usually achieved through developing a work breakdown structure, a schedule for its completion and reintegration, plans of how the work is to be completed, standards of how the work is to be done and standards relating to what work needs to be produced [27-29]. Since software development is inherently uncertain, there is usually a need for dynamic coordination, achieved through meetings, reviews and other exchanges. As with performance, the method needs to provide for coordination and, furthermore, it needs should be adequate for the range of circumstances encountered.

3 Method Enactment

The enactment assessment focusses on the SME-*constructed* process but may identify the need to replace or augment the method fragments. The artifact under assessment is the enacted method.

When projects are being planned, it is with considerable knowledge of the expected project contingencies and constraints. Some constraints are fixed, like the delivery date of software required for a specific event. Some are negotiable, like the number of people in the development team or the actual scope of the project. While such constraints are normally considered as part of project planning, they also contribute to method tailoring during that planning. Method enactment describes the association of parameters in the designed method to actual project-specific resources, such as actual team members, real deadlines, available funding etc. and the subsequent method tailoring.

There is a considerable body of information on contingency theory relating to software development of which some is discussed in the paragraphs that follow. However, there is little consensus on the contingencies of importance in different circumstances. Additionally, other fields such as product development [30] have the potential to open discussion on method enactment to a much wider treatment than the current concerns of software project planning. For these reasons, this paper will not attempt a rigorous examination of specific contingency factors that may impact software development projects.

Project contingencies such as the size of the project are generally recognized as influencing the choice of method [31, 32]. Because larger projects generally involve more people, coordination tends to be more mechanistic [28, 33] than the more organic methods typical of smaller projects. Expressed differently, larger projects tend to use plans, standards and formal exchanges to coordinate their work whilst smaller projects tend to use stand-up meetings or co-location.

Many software development methods do not yet address the effect of a distributed development team. An exception are some of the practices incorporated into the Crystal family of communicating between team members. These are significant in agile development methods [17] where the problems of communicating between team members is acknowledged and different techniques are proposed to overcome barriers to communication. Other less agile development methods seem to be unaffected by team distribution implying that the projects concerned were already using techniques that were less affected by distance [34]. It is also possible that some methods are more susceptible to communication barriers than others and that compensation is sometimes possible.

Various authors have identified sources of uncertainty including platform and market uncertainty [35], requirements uncertainty [28], outcome uncertainty and task programmability [36] as significantly influencing choices of method and its tailoring. While specific relationships between different types of uncertainty and method may need some investigation, the general connection seems to be well accepted.

Safety critical or security critical applications may demand extra processes (in the ISO sense i.e. a second meaning to the one depicted in Figure 2), intended to augment an otherwise less critical method [37, 38]. In this case, the contingency is addressed by extra processes and activities rather than a selection among equivalent activities in

a process already included in the method. A project with stringent quality requirements will need to meet those requirements through more rigorous verification activities which may, in turn, affect the selection of personnel on the verification team and the distribution of other tasks. Method enactment involves more than mere adjustment to parts of the method. It may require wholesale redistribution of activities within the method.

Although the project management literature does not specifically identify such project planning activities as method tailoring, it is achieved nonetheless. For example, a project manager may be faced with a demand to outsource some of the development, leading to a question of where verification of the outsourced development is to be performed. Verification of the work could be performed by the developers, if they were known and trusted, or by the acquirer after delivery. Such decisions will be reflected in the method as different ways in which activities are grouped together to form processes and the allocation of those processes to organizations.

The main contingencies of software development projects, described above, are those that are normally considered when a project is planned. In the parlance of method engineering, the method is configured. Method configuration through the selection of specific activities suited to the circumstances has been discussed in [9, 10].

In contrast to assessment at method design, method assessment at enactment has specific information about the project constraints and contingencies so can be expected to determine the utility of the method with greater certainty than previously possible. Assessment at enactment is unlikely to be significantly different from assessment at design but has greater immediacy. An assessment would be expected to determine whether or not the proposed means of monitoring and managing performance is likely to work in the specific circumstances. For example, if the project is large and distributed, oversight of the distributed organizations cannot rely on the same oversight processes as would be employed if all parties belonged to the same organization. Similarly, coordination processes at the low range of project size, where co-location may be possible, would be different from coordination processes needed at the high end of project size.

A method assessment at enactment would help avoid a tendency to use a familiar but possibly inappropriate method. It would help direct attention to parts of the method affected by the constraints and contingencies and help remove subject judgement about whether or not the method "feels" right.

4 Method Performance

The whole point about tailoring a method for specific constraints and contingencies is to achieve the best possible outcome in practice. A tailored method represents a hypothesis that the best outcome possible will be achieved under the specific constraints and contingencies. Like all hypotheses, it should be possible to gather evidence to prove or disprove it. Yet so far there seems to have been little attempt to do so. An assessment should be able to identify two distinct issues: the right method but the wrong performance and the wrong method with the right performance. An assumption that the first case is, by default, true seems to dominate existing process assessment methods [39, 40] and also seems to underlie quality management methods [41].

However, the argument of situational method engineering is that the method may possibly be unsuited to the circumstances leading to poor outcomes no matter how the method is performed.

Performance is usually negotiated during project planning as the scope of the project, the available personnel, quality and other requirements, budget and delivery milestones are considered and resolved [42-44]. Performance monitoring and management would normally be achieved through some unspecified means, e.g. project review meetings, to inform the project of changes in the constraints and a means to collect and report performance data. A method assessment would need to examine whether these activities were present and adequate for the expected range of circumstances.

Available process assessment methods such as SPICE [13] and CMMI [14] are assessments of process capability, not of performance. Moreover, they assess processes and not the overall performance of the method. Process capability attempts to measure the degree to which a given process is likely to achieve its stated purpose [45]. By itself, that says nothing about how well suited the process is to the circumstances. That determination relies on the knowledge and experience of the process assessor to decide whether the process itself is flawed and needs improvement or whether the process is being incorrectly performed for one reason or another. Other forms of assessment, such as a process audit, generally rely on the judgement of the auditor. This dependence on the well-intended judgement of an experienced auditor or assessor is unsatisfactory for a number of reasons. The first is that the assessor may not be sufficiently knowledgeable about software development or the particular circumstances, leading to well intended but harmful findings. The second is that such assessors tend to be expensive, out of reach for any but large software developers with project budgets able to absorb the high cost of a rigorous assessment.

Informal assessments are done all the time. People learn and adjust what they do. For example, Scrum practice includes a retrospective at the end of each sprint [20, 46] and most methods have some sort of post-mortem or other form of audit; the need for continual improvement is built into ISO 9001. However, these general imperatives to improve don't provide guidance on what to look for or how to recognize the need for improvements.

During method performance, information is available about how well the method is achieving its intended purpose. It is necessary to compare actual performance to expected performance in order to determine where changes to the method (e.g. in the case of the wrong process performed correctly) are required or where changes to method performance (the right process performed incorrectly) are needed.

However, it is apparent that most of the method attributes of method performance, the quality of both the fragments and the constructed method are evaluated in real time. Feedback may well suggest a dynamic replacement or the addition of new method fragments in order to ensure that the project is successfully completed. Of especial interest are compound attributes for which there are no direct measures. Nor do there seem to be generally agreed-upon models of effectiveness, efficiency, coordination or governance. Like project success, it may be difficult to say what is required to achieve it, but relatively easy to determine if it is not being achieved. Rather than try to show that these are present and being achieved, it should be possible to detect their absence through errors or other symptoms of failure.

Method performance assessment would prompt a review of the project constraints and contingencies. Conversely, a change in project contingencies or constraints may prompt a method performance assessment. Additionally, a method performance assessment would provide an additional means to detect and justify changes to the method or to the manner of performing the method. It is not proposed that method performance assessment is equivalent to or should replace other, more rigorous, process assessments. The proposed method performance assessment is intended to directly and objectively assess the appropriateness of the method rather than imply it through the subjective knowledge and experience of a process assessor.

During assessment of method performance, the quality of both the fragments and the constructed method are evaluated in real time. Feedback may well suggest a dynamic replacement or the addition of new method fragments in order to ensure that the project is successfully completed.

5 Advantages of This Framework

This proposed framework identifies method design, method enactment and method performance separating the concerns of each phase and identifying what can be reasonably achieved at each phase. The proposed framework also links the typical activities of projects, particularly software development projects, to necessary changes in methods, drawing attention to different situational method engineering activities that might be necessary.

The framework provides guidance about what could be assessed at each phase as well as the desirability of doing so. This may assist method engineers to assess their methods more rigorously than through the more basic and more subjective means of expert opinion – as is currently the case. Method assessment techniques may be developed that can assist non-specialists review their intended method or method performance without needing to resort to expensive audits or formal process assessments. Given that most of the world's software developers claim that they do not use a formal software development method [47, 48] and certainly don't review or assess what method they do use, a more readily usable method assessment technique would seem to offer some advantages.

The proposed framework provides a guide to development of assessment techniques and tools. Rather than try to develop a general assessment technique that attempts to require certainty where none is possible (method design) or fails to require rigor where some is possible (method enactment), tools can be developed to assess methods appropriately and as rigorously as possible, but no more. Similarly, existing assessment techniques can be positioned in relation to the type of method assessment they accomplish.

6 Discussion and Related Work

There has been very little direct research on this topic, However, Perez et al. [49] discuss congruence evaluation, arguing that this is the most important measure to be assessed. Congruence is proposed as a measure of how well the process model fits the

intended usage/domain. They suggest the use of a contingency model as a precursor to defining the relationships between the process model and its intended usage context. They introduce three variables: a dependent variable (the effectiveness of the process model), an independent variable (a characteristic of the process model) and a contingency variable (either a characteristic of the context or the process model). They recommend that attribute values should first be assigned, often subjectively by someone familiar with the situation, and then a congruence measure calculated based on the inter-relationships established between the process model and the attributes of the process context, the derived congruence index being a real number in the closed interval (-1,1). Here, 1 indicates a perfect match and -1 the worst possible match. Low congruence values can thus be identified as suggesting that improvement is needed. A similar, contingency approach is also advocated by [50] (based on work of [51]). Using a banking example, these authors recommend the following list of contingency factors:

- Management commitment,
- Importance,
- Impact,
- Resistance and conflict,
- Time pressure,
- Shortage of human resources,
- Shortage of means,
- Formality,
- Knowledge and experience,
- Skills,
- Size,
- Relationships,
- Dependency,
- Clarity,
- Stability,
- Complexity,
- Level of innovation.

Adaptation of processes is also discussed by [52] in the context of agile methodologies.

In a much broader software development context, Kherrai et al. [53] discuss the need to use a set of quality attributes from ISO 9126 [54] stressing the need to do this for the strategy and tactics of the software project plan. A different focus is taken by [55] in which the concerns of governance, risk and compliance of the to-be-constructed method are highlighted.

Although not discussing assessment per se, Niknafs and Asadi [56], in the context of CAME (computer-aided method engineering), do split the process into four stages: enactment, elicitation, evolution and evaluation and focus on static descriptors (that they call aspects) rather than a time-based characterization as we discuss here. Key notions for method adaptation are examined in [57] although, once again, there is no discussion regarding how this might be incorporated into process assessment.

7 Conclusion

We have proposed here a framework to enable assessment of methods at different times in their life cycle. The purpose of the framework is to enable method engineers to assess the congruence of a developed, enacted or performed method according to the claimed range of known constraints and contingencies. In describing the proposed framework, we have shown that there is much to be gained from separating method assessment into the different types so that the method can be assessed appropriately and useful information provided to stakeholders. The framework and assessment techniques and metrics that may be developed in the future to support it offers the possibility that tailoring methods appropriately might be within the reach of more and smaller organizations.

The framework highlights some fruitful areas for research. In particular, it underlines the need to better link actual software development practices and their tailoring to situational method engineering. The need to coordinate work cuts across processes, as does process performance monitoring and management, and their interaction with the primary production processes could be explored. Measures of method attributes and techniques of method assessment are also identified by the framework. Research arising from the framework promises to be highly relevant to industry.

Acknowledgements

This is paper number 10/10 of the Centre for Object Technology Applications and Research within the Centre for Human Centred Technology Design of the University of Technology, Sydney.

References

1. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information and Software Technology 38(4), 275–280 (1996)
2. Henderson-Sellers, B., Ralyte, J.: Situational Method Engineering: A state of the art review. Journal of Universal Computer Science 16(3) (2010)
3. Kumar, K., Welke, R.J.: Methodology Engineering: a proposal for situation-specific methodology construction. In: Challenges and Strategies For Research in Systems Development, pp. 257–269. John Wiley & Sons, Inc., Chichester (1992)
4. Ågerfalk, P., Brinkkemper, S., Gonzalez-Perez, C., Henderson-Sellers, B., Karlsson, F., Kelly, S., Ralyté, J.: Modularization Constructs in Method Engineering: Towards Common Ground? In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences, vol. 244, pp. 359–368. Springer, Boston (2007)
5. Gonzalez-Perez, C., Henderson-Sellers, B.: Metamodelling for Software Engineering. Wiley Publishing, Chichester (2008)
6. Henderson-Sellers, B.: Method engineering for OO systems development, vol. 46, pp. 73–78. ACM, New York (2003)

7. Karlsson, F., Ågerfalk, P.J.: Method configuration: adapting to situational characteristics while creating reusable assets. Information and Software Technology 46(9), 619–633 (2004)
8. Kornyshova, E., Deneckère, R., Salinesi, C.: Method Chunks Selection by Multicriteria Techniques: an Extension of the Assembly-based Approach. In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences, vol. 244, pp. 64–78. Springer, Boston (2007)
9. Henderson-Sellers, B., Nguyen, V.P.: Un outil d'aide à l'ingénierie de méthodes reposant sur l'approche OPEN. Génie Logiciel (70), 12 (2004)
10. Ralyté, J., Rolland, C.: An Approach for Method Reengineering. In: Kunii, H.S., Jajodia, S., Sølvberg, A. (eds.) ER 2001. LNCS, vol. 2224, pp. 471–484. Springer, Heidelberg (2001)
11. DeLoach, S., Garcia-Ojeda: O-MaSE: An Customizable Approach to Designing and Building Complex, Adaptive Multiagent Systems. International Journal of Agent-Oriented Software Engineering 4(3) (in Press)
12. Henderson-Sellers, B.: Method Engineering: Theory and Practice. In: Information Systems Technology and Its Applications. In: 5th International Conference ISTA, p. 84. Gesellschaft Für Informatik (2006)
13. ISO/IEC 15504-1:2004 - Information Technology - Process Assessment - Part 1: Concepts and Vocabulary
14. SEI. CMMI® for Development, Version 1.2. CMU/SEI-2006-TR-008 (2006)
15. Royce, W.W.: Managing the development of large software systems: concepts and techniques. In: Proceedings of IEEE WESCON. IEEE Computer Society Press, Monterey (1970)
16. Boehm, B.W.: A spiral model of software development and enhancement. Computer 21(5), 61–72 (1988)
17. Cockburn, A.: Agile Software Development: The Cooperative Game, 2nd edn. Addison-Wesley Professional, Reading (2006)
18. Beck, K.: Extreme Programming Explained. Addison-Wesley, Boston (2000)
19. Highsmith, J., Cockburn, A.: Agile software development: the business of innovation. Computer 34(9), 120–127 (2001)
20. Sutherland, J.: Scrum software development process. In: OOPSLA (1995)
21. Bajec, M., Vavpotic, D., Krisper, M.: Practice-driven approach for creating project-specific software development methods. Information and Software Technology 49(4), 345–365 (2007)
22. Graham, I.: Object-Oriented Methods: Principles and Practice, 3rd edn. Addison-Wesley Professional, Reading (2000)
23. Rolland, C.: Method engineering: Towards Methods as Services. Software Process: Improvement and Practice 14, 143–164 (2009)
24. Ralyté, J., Rolland, C.: An Assembly Process Model for Method Engineering. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 267–283. Springer, Heidelberg (2001)
25. Ralyté, J., Deneckère, R., Rolland, C.: Towards a Generic Model for Situational Method Engineering. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 1029–1029. Springer, Heidelberg (2003)
26. Goldkuhl, G., Lind, M.: Coordination and transformation in business processes: Towards an integrated view. Business Process Management Journal (14), 761–777 (2008)
27. Kraut, R.E., Streeter, L.A.: Coordination in software development. Communications of the ACM 38(3), 69–81 (1995)

28. Nidumolu, S.R.: A comparison of the structural contingency and risk-based perspectives on coordination in software development projects. Journal of Management Information Systems 13(2), 77–113 (1996)
29. McBride, T.: The mechanisms of project management of software development. Journal of Systems and Software 81(12), 2386–2395 (2008)
30. Reinertsen, D.G.: The Principles of Product Development Flow: Second Generation Lean Product Development. Celeritas Publishing, Redondo Beach (2009)
31. Cockburn, A. (2004), http://alistair.cockburn.us/crystal/crystal.html
32. Elssamadisy, A., Schalliol, G.: Recognizing and responding to "bad smells" in extreme programming. In: 24rd International Conference on Software Engineering, pp. 617–622 (2002)
33. Andres, H.P., Zmud, R.W.: A Contingency Approach to Software Project Coordination. Journal of Management Information Systems 18(3), 41–70 (2002)
34. McBride, T., Henderson-Sellers, B., Zowghi, D.: Managing outsourced software development: Does organisational distance demand different project management? In: UKAIS 2006 (2006)
35. MacCormack, A., Verganti, R.: Managing the Sources of Uncertainty: Matching Process and Context in Software Development. Journal of Product Innovation Management 20(3), 217–232 (2003)
36. Eisenhardt, K.M.: Agency Theory: An Assessment and Review. Academy of Management Review 14(1), 57–74 (1989)
37. Davis, N.: Secure Software Development Life Cycle Processes: A Technology Scouting Report. CMU/SEI-2005-TN-024 (2005)
38. ISO/IEC 15504-10:2010 - Information technology — Process assessment — Part 10: Safety extension
39. SEI. Standard CMMI Appraisal Method for Process Improvement (SCAMPI). CMU/SEI-2001-HB-001 (2001)
40. ISO/IEC 15504-3:2004 - Information technology - Process assessment - Part 3: Guidance on performing an assessment
41. ISO/IEC 9001:2000 - Quality Management Systems - Requirements
42. Burke, R.: Project Management: Planning and Control Techniques. Burke Publishing, Tokai (2003)
43. Cleland, D.I., Ireland, L.R.: Project management: Strategic Design and Implementation. McGraw-Hill, New York (2002)
44. Hughes, B., Cotterell, M.: Software Project Management. McGraw-Hill, London (1999)
45. ISO/IEC 15504-2:2004 - Information technology - Software process assessment - A reference model for processes and process capability
46. Rising, L., Janoff, N.S.: The Scrum software development process for small teams. IEEE Software 17(4), 26–32 (2000)
47. Fitzgerald, B.: The use of systems development methodologies in practice: a field study. Information Systems Journal 7(3), 201–212 (1997)
48. SEI. Process Maturity Profile - CMMI 2005 Year-end Update (2005)
49. Pérez, G., El Emam, K., Madhavji, N.: Customising software process models. In: Schäfer, W. (ed.) EWSPT 1995. LNCS, vol. 913, pp. 70–78. Springer, Heidelberg (1995)
50. van Slooten, K., Hodes, B.: Proceedings of IFIP TC8 Working Conference on Method Engineering: Principles of Method Construction and Tool Support, pp. 29–44. Chapman and Hall, Boca Raton (1996)
51. van de Hoef, R., Harmsen, A.F., Wijers, G.M.: Situatie, Scenario & Succes (1995)

52. Henninger, S., Ivaturi, A., Nuli, K., Thirunavukkaras, A.: Supporting Adaptable Methodologies to Meet Evolving Project Needs. In: Wells, D., Williams, L. (eds.) XP 2002. LNCS, vol. 2418, pp. 33–51. Springer, Heidelberg (2002)
53. Kherraf, S., Cheikhi, L., Abran, A., Suryn, W., Lefebvre, E.: The Need to Evaluate Strategy and Tactics before the Software Development Process Begins. Journal of Software Engineering and Applications 3(7), 644 (2010)
54. ISO/IEC 9126:2001 - Software engineering - Product quality - Part 1: Quality model
55. Gericke, A., Fill, H.-G., Karagiannis, D., Winter, R.: Situational method engineering for governance, risk and compliance information systems. In: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology. ACM, Philadelphia (2009)
56. Niknafs, A., Asadi, M.: Towards a Process Modeling Language for Method Engineering Support. In: WRI World Congress on Computer Science and Information Engineering, pp. 674–681 (2009)
57. Aydin, M.: Examining Key Notions for Method Adaptation. In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences, vol. 244, pp. 49–63. Springer, Boston (2007)

Towards Common Ground in SME: An Ontology of Method Descriptors

Adrian Iacovelli and Carine Souveyet

University Panthéon Sorbonne. 90 rue Tolbiac,
75013 Paris, France
{adrian.iacovelli,souveyet}@univ-paris1.fr

Abstract. The Method Engineering (ME) community is a prolific research domain where competing Situational Method Engineering (SME) approaches have been defined and used for composing, adapting or/and configuring a method into modular constructs according to their own modularization vision. This diversity shows the richness of the ME domain but implies some drawback like unnecessary confusion for non ME expert, lack of standard & interoperability, lack of implementation tool. However, researchers are agreed that a common ground in SME is a hot matter of discussion. Assuming that the differences between SME approaches are purposeful, we propose to reach a semantic common ground on what types of core concepts constitute a method descriptor. To achieve it, an ontology-based approach is applied in SME to design an ontology of method descriptors as a domain ontology. The semantics of the six most popular SME approaches modular constructs are defined according to this ontology in order to show its usage and its relevance. Finally, usage scenarios have been sketched to show that the ontology can be the start up phase for reducing the ME drawbacks mentioned above.

Keywords: Method Engineering, Method Descriptors, Ontology, Service Oriented Architecture.

1 Introduction

Information systems development methods are the subject of study of Method Engineering (ME) science. One of the ME interests is to decompose into modular parts these methods for optimizing, reusing, and ensuring their flexibility and their adaptability [1]. This interest is the basis of the Situational Method Engineering (SME) community. This domain is a prolific research domain where several competing SME approaches have been defined, published and used with their own vision of method modularization. This diversity shows the richness of the research works but implies some drawbacks like unnecessary confusion for non ME experts [1], lack of standard & interoperability, lack of implementation tool [2].

Today, a common ground in ME is a hot topic of discussion between researchers [1]. According to the authors of [1], there are two possible solutions: (1) differences are minor and an agreement on what modular construct to promote can be reached, or (2) the diversity is useful because they serve different purposes and there is a need for them to co-exist.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 77–90, 2011.

In the past, we had published a framework for the method modular constructs comparison for underlying their semantic differences [2, 3], and pushed our vision for a specific matter. However, today, we believe and assume that the diversity is purposeful. But we also believe that a semantic common ground in SME is needed and can be achieved. This semantic common ground can be considered as a start-up phase to reduce directly or indirectly drawbacks such as (a) unnecessary confusion for non ME expert, (b) lack of standard & interoperability, and (c) lack of implementation tool.

The purpose of this paper is to propose an ontology-based approach to design the semantic common ground in SME and sketches its benefits into exploring scenarios to reduce the drawbacks mentioned above. An ontology was proposed in the ME field in [4] to define the core concepts required for qualifying knowledge about method. But it is a top lightweight ontology which not allows to define a common ground for SME approaches. Another ME based ontology was proposed in [5] but their concepts are too restrictive to cover the diversity of method modular constructs and levels of granularity introduced in the various SME approaches. In addition, their objective is to improve the SME approach proposed in [6] and not to find a common ground in SME.

In a philosophical point of view, ontology is the study of the categories of things that exist or may exist in a particular domain. In other words, domain ontology defines the types of things in that domain. Moreover, ontology is a fundamental part of the knowledge, and all other knowledge should rely on it or refer to it.

SME approaches [7, 8, 9, 11, 11] promote different modular constructs of a method but they have a common understanding of what a method is [12, 13]. Here, a method is described by five interrelated ways: a way of thinking (paradigm), a way of modelling (product), a way of working (process), a way of supporting (tool) and a way of controlling (organisation). Therefore, the core of ME is represented by the common understanding of the various things that constitute or may constitute a method description. Consequently, method descriptors ontology is designed as domain ontology and the various modular constructs of SME approaches such as 'method fragment', 'method chunk', 'method component', 'process component' are defined semantically by referring the concepts of the ontology.

The paper is organised as follows. The ontology of method descriptors is explained in section 2. Section 3 illustrates how various SME modular constructs match the ontology. Furthermore, exploring usage scenarios of the Ontology are sketched in section 4 to illustrate the interest of the ontology and to explore future research options for reducing the ME drawbacks mentioned above. Finally, section 5 concludes this work with our contribution and research perspectives.

2 Method Descriptors Ontology

This section explains the concepts and their relationships defined in the Ontology of method descriptors. The SME approaches have different method modular constructs but they agree on the understanding of what a method is [12, 13]. According to this definition, a method is designed as a collection of method modular constructs. Figure 1 illustrates our ontology is built upon this definition.

The reminder of this section explains the concepts defined in the Ontology of Method descriptors and their relationships.

The following top concepts of the ontology illustrated in Figure 1 are rooted to the "*Thing*" concept : *Method Puzzle*, *Goal*, *Verb*, *Target*, *Parameters*, *Paradigm*, *Concept*, *Modelling Element*, *Modelling Rule*, *Annotation*. Moreover, "is a" links of the ontology define specialisation relationships between two concepts.

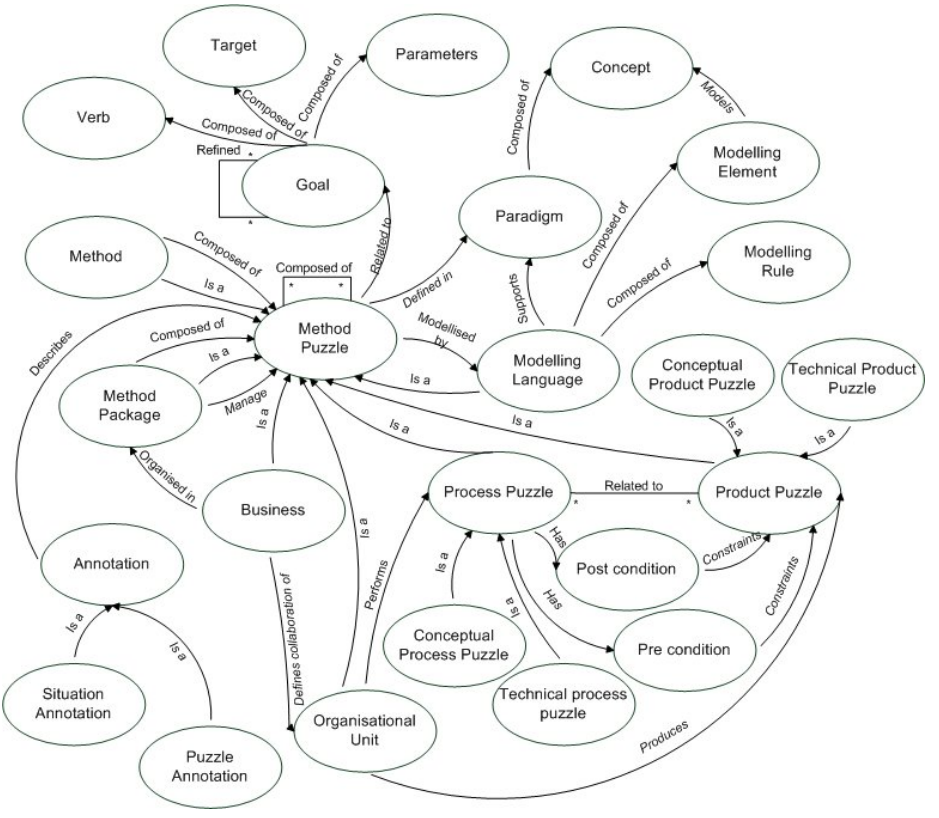


Fig. 1. Method Descriptors Ontology

The word *method* comes from the Greek *methodos* which means way of investigation [7]. According to Harmsen a method is an integrated collection of procedures, techniques, product descriptions and tools for an effective and efficient support on the engineering process [14]. In the situational method engineering context, the method modular constructs (i.e. *Method Puzzle*) are assembled to produce a method tuned to its application situation. This decomposition into modular *Method Puzzle* parts is fundamental for the flexibility, adaptability, optimization and reuse of methods [15]. As a composition of *Method Puzzle*, a *Method* is also viewed as a *Method Puzzle*.

A *Method Package* is an autonomous reusable part of a *Method* capturing one of its particular aspects. It defines an assembly of *Method Puzzles* responding to a specific

kind of project. It is either a preconfigured part of a *Method.* Alternatively, it can incorporate a temporal dimension to organize its composing *Method Puzzles* [8, 16, 17].

A *Method Puzzle* describes an element of a method. It's a coherent piece of an IS development method [7]. In order to manage the complexity of a method definition, SME approaches emphasize a modular vision of its definition. This vision is introduced by the *Method Puzzle* concept. *Method Puzzles* can be defined at different level of granularity, i.e. *Method Puzzles* can be composed of other *Method Puzzles*. In addition a *Method Package* is viewed as a composition of *Method Puzzles* and is also considered as a *Method Puzzle*. Notice that the two specializations of a method puzzle: method and method package are exclusive with all other specializations. A *Method Puzzle* specialized into a *Method* or a *Method Package* can't have any other specialization. The other *Method Puzzle* specializations correspond to the Seligmann and Rolland [12, 13] method definition: *way of thinking, a way of modeling, a way of working, a way of control and a way of supporting*. The way of thinking is the philosophy used in the *Method Puzzle* which is captured in the *Paradigm* concept and supported by a *Modeling Language*. The way of modeling describes the various constructs and their models related to the method application. This way is defined into the *Product Puzzle* concept and more precisely by the *Conceptual Product Puzzle* concept. To complete the way modeling definition, the way of working expresses how to perform a method or how a product evolves during a method. This way is identified in our ontology by the *Process Puzzle* concept and more particularly by the *Conceptual Process Puzzle* concept. The way of control specifies how to organize the performing of a method process into an organization and is described by the *Business* and *Organizational Unit* concepts. The way of supporting is the tool for supporting the method. It is related to the *Technical Process Puzzle* and to the *Technical Product Puzzle* concepts. The various SME approach share the same view of a *Method* but have different definitions of a *Method Puzzle*. To take into account this diversity, the various specialization of *Method Puzzle* such as *Product Puzzle, Process Puzzle, Modeling Language, Business* and *Organizational Unit* are inclusive. For example a *Method Puzzle* can be specialized into both *Process* and *Product Puzzles* at the same time like in the Chunk Approach [11].

The main purpose of a method puzzle is to be reused in different methods. In order to increase its reusability, one has to provide a mechanism for extracting and regrouping the key concepts of a method puzzle. This concise information on a method puzzle is called annotation. It helps searching and retrieving method puzzle. Two types of information are required: (a) information regarding the situations where a *Method Puzzle* can be reused and (b) information to characterize and summarize the content of a *Method Puzzle*. The first is managed by the *Situation Annotation* and the second is handled by the *Puzzle Annotation*.

A *Method Puzzle* helps to achieve a particular *Goal*. A Goal in this case is a statement expressing what is wanted [18]. I.e. it represents the state to be reached or maintained. A linguistic approach proposed in [19] and its extension in [20] are based on the case grammar. They recognize a goal statement as a combination of a verb, a target and parameters. A goal verb is the central component of the statement. It describes the action to be performed. The target is the subject of a goal statement. It can depict the expected result of a goal achievement or an existing entity modified by performing a goal. In addition parameters are complementary information exposed in

a goal statement. In fact, each parameter plays a semantic role according to the verb. Goals can be defined at different levels of granularity. It means that the achievement of a complex goal may require the achievement of sub-goals. We say here that a goal can be refined by a set of (sub) goals. In the ontology it is expressed by the recursive *refined* relationship of the goal concept.

Paradigm is a coherent model of a world perception grounded on a specific philosophy. A *Paradigm* describes a set of concepts and their interactions that cannot be mixed with another *Paradigm*. In our ontology, concepts used to express the paradigm are introduced by the *Concept* node**.** This node helps to represent the constructs and their relationships useful for the paradigm description. A Modeling Language is used to design world according to the concepts of the paradigm. It is considered as a tool to produce models. A Modeling Language can be itself defined as a *Method Puzzle*: a set of *Modeling Elements* defined according to *Modeling Rules*. A *Modeling Element* is a textual or graphical representation of a *Concept* from a *Paradigm* whereas a *Modeling Rule* is an axiom that must be satisfied by modeling elements or a set of *Modeling Elements.*

In SME approaches, the *Process Puzzle* is one of the key concepts. It is an abstract element aimed to capture a process for achieving the *Method Puzzle Goal* [10]. It is the work that has to be done in order to obtain the result [17]. Or, more precisely, it is the set of actions which transforms a product (*Product Puzzle*) under development [11], from a source product to a target product [8]. As a *Process Puzzle* is a specialization of a Method Puzzle, it can be defined at various level of granularity. So it can describe high-level project strategies or more detailed development procedures [7]. The modeling of this process structure is supported by the *Conceptual Process Puzzle* concept. This concept includes a set of process descriptions and models. Its implementation is supported by the *Technical Process Puzzle* concept. This concept represents an operational tool automation of the *Process Puzzle*. As shown in [10], two other concepts are related to the *Process Puzzle*: a *Precondition* and a *Postcondition*. The Precondition concept defines an initial situation required for applying a *Process Puzzle*. It is a restriction constraining the input *Product Puzzles* instances of a *Process Puzzle*. A *Precondition* defines the expected state of *Process Puzzle* input products. At the opposite, a *Postcondition* concept defines a final situation resulting of the application of a *Process Puzzle*. It is a restriction constraining the output *Product Puzzles* instances after the performing of a *Process Puzzle*. The *Postcondition* defines the expected state of Process Puzzle output products.

Another key concept of SME approaches is the *Product Puzzle*. It's an abstract element capturing the product aspect of methodologies [17] and it conforms to the paradigm adopted in the methodologies. A Product Puzzle models artifacts used or produced by the performing of a Process Puzzle [7, 17, 10, 11]. These artifact models are defined as *Conceptual Product Puzzle* concept whereas their instance implementations are supported by *Technical Product Puzzle* concept.

The way of control identified above is represented by the *Business* and *Organizational unit* concepts. An *Organizational Unit* is a resource, an actor role or a set of actors (team or bigger groups) description involved into performing of a *Process Puzzle* in order to produce a product described in a *Product Puzzle*. A *Business* is used to model the collaboration of *Organizational Units*. It captures the interactions between *Organizational Units* in order to perform a project or a business mission of

an enterprise [21]. A Business is related to several *Method Packages* and temporally organized into them. The business concept can be also considered as a Method puzzle.

In this section the core concepts of a SME descriptor that constitute the ontology have been explained. The next section illustrates how these concepts are used to define the semantics of modular constructs of selected SME approaches.

3 SME Method Descriptors

This section illustrates how our ontology defines semantics for each SME modular construct. We have selected the five most cited component-based SME approaches such as [7, 8, 9, 10, 11] and one approach defined in the service orientation. We show that the ontology can constitute a common ground in SME approaches which are component or service based approaches.

3.1 Method Fragments

In [7], Brinkkemper and colleagues propose the method fragment concept. This concept is one of the earliest modularization constructs in ME [1]. According to [7] method fragment is a standardized building block based on a coherent part of a method [7]. It is an abstract element defined at one of the five different layers of granularity: method, stage, model, diagram, or concept [22]. The method fragment concept matches with the *method puzzle* concept as it is defined in our ontology and its granularity is captured by the composition link. A fragment can be specialized either into a process fragment or a product fragment (cf. Figure 2). As product fragments models the structures of the methods products and process fragments are models of the development process [7], they can be respectively defined as a specialisation of the *conceptual product and process* puzzles concepts.

This specialisation is a specific case of the ontology method puzzle as it's an exclusive specialisation into product or process fragments. The retrieval and use of

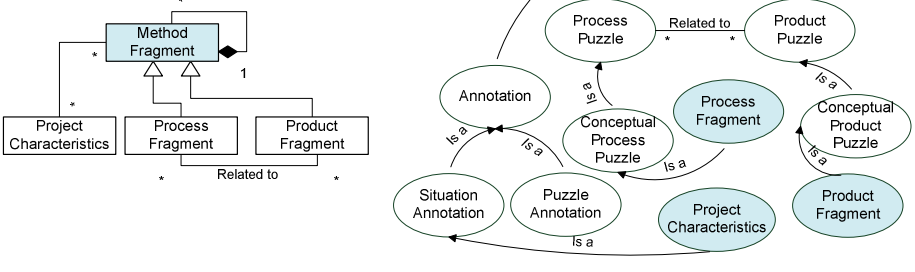


Fig. 2. Method Fragment Structure and Semantic Matching

method fragments is provided by project characteristics attached to each fragments. These situation characteristics match with the situation annotation concept of the ontology.

3.2 Method Chunks

The method chunk approach was proposed by Rolland and colleagues [11]. A method chunk is organised into two levels of knowledge: a method knowledge level and a meta-knowledge level [1, 23]. The method level of the method chunk concept is driven by its method process part which is attached the product part. As a method chunk is a composition of one process part and one product part, the method chunk is characterized by an inclusive specialisation into both a *conceptual process puzzle* and the *conceptual product puzzle*. The process and product part are defined respectively in our ontology as a specialisation of the *conceptual process puzzle* and the *conceptual product puzzle* with a more specific one to one cardinality on the relationship between their *process puzzle* and *product puzzle* parent concepts. The body concept in the method chunk approach is an abstract concept design for the encapsulation the process and product part. As it's an abstract concept that doesn't support additional new semantic to the chunk concept, there is no need to model it in the ontology. The interface of a method chunk captures information on the chunk and its goal. The matching of the interface concept in our ontology is done by a double inheritance link with the *puzzle annotation* and the *goal* concepts. In the same way, the meta-knowledge level of method chunks captured by the descriptor concept is defined with a double inheritance link the *situation annotation* and the *goal* concepts.

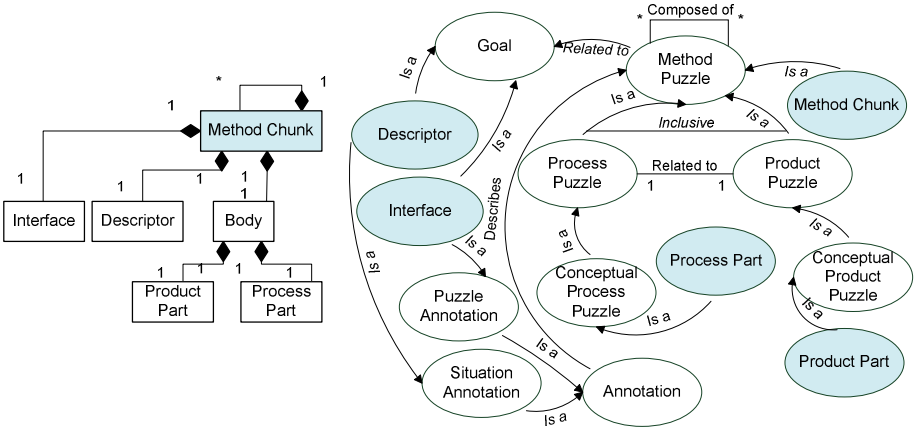


Fig. 3. Method Chunk Structure

The purpose of the descriptor concept is to capture the situational aspects of method chunks usage to support their retrieval process. Descriptor contains the goal definition of the chunk and a set of parameters characterising the situation of reuse of this chunk.

3.3 Method Components

The method component concept aims to capture a self-contained part of a system engineering method [8]. The latest contributions to this concept were made by Karlsson and colleagues in [8, 16]. Method component is designed to be used in a specific kind of ME, the method configuration. Each component has to address a certain aspect of the problem at hand and it is the smallest part of a method that is practically useful [1]. For these reasons the method component concept is mapped to the method package concept of our ontology. A method component is build by an assembly of several method elements that are the basic constructs constituent of a method: action, artefact, actor role, concept and notation. This method element concept can be defined as a specialisation of the method puzzle concept. An action is the set of tasks to be performed during the method component application. As actions are the central constituents of the method process model [8] they can be defined as a specialisation of the *conceptual process puzzle concept* in our ontology. The results of these actions are represented by artefacts in the method product model [8]. The artefact concept is matched with the conceptual *product puzzle concept*. Furthermore, the actions are performed by project members who have different roles during the project [8], this implies that the actor role concept can be mapped with the organisational unit concept of the ontology. A set of concepts is used to describe the problem domain of the method component and they are captured and represented using notations [8]. These concepts respectively correspond to the *concept* concept and the *modelling element* concept of the ontology. Both method components and method elements are linked to goals which can be refined in sub-goals. That defines a perfect match between the goal concept form the method component approach with the goal concept of the ontology.

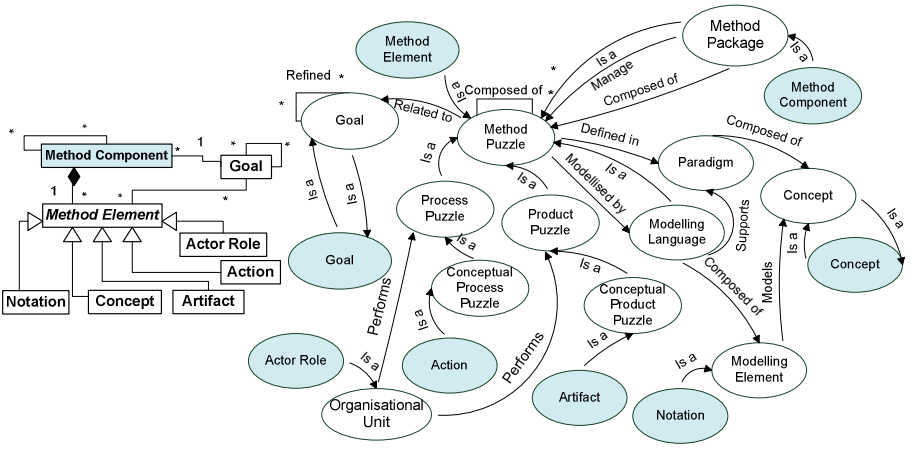


Fig. 4. Method Component Structure

3.4 Open Process Framework (OPF) Method Elements

Based on the international standard ISO/IEC 24744 [24] the OPF approach was proposed by Henderson-Sellers and colleagues [9], the last updates of the approach can

be found in [21]. Each OPF method component is generated from an element in a prescribed underpinning meta-model [1] according to the ISO standard. An OPF method component is defined as an abstract element which all other method constructs are derived [21]. So it can be defined as a specialisation of the *method puzzle* concept in our ontology. The OPF approach is driven by the decomposition of the method process in work units performed by producers known as people role and teams. These two latter concepts match respectively with the *conceptual process puzzle* and the *organisational unit* concepts of our ontology. Various products are used or created by work units in order to deliver the final system [17]. This product aspect of methodologies is captured in the work product concept of the OPF approach that can be defined as a specialisation of the *conceptual product puzzle* of the ontology. The work products are documented using a language consisting in a "vocabulary" and a set of "grammatical rules" [21]. These latter concepts can be mapped respectively with our *modelling language*, *modelling element* and *modelling rule* concepts. All these OPF method components are used during a stage and performed by a specific collaboration organisation of producers called a endeavour [21]. A stage models the intended timing of the performance of a temporally-cohesive set of work units during the enactment of a method [21] and can be defined as a specialisation of the ontology *method package* concept whereas the endeavour concept can be defined as a specialisation of the *business* concept.

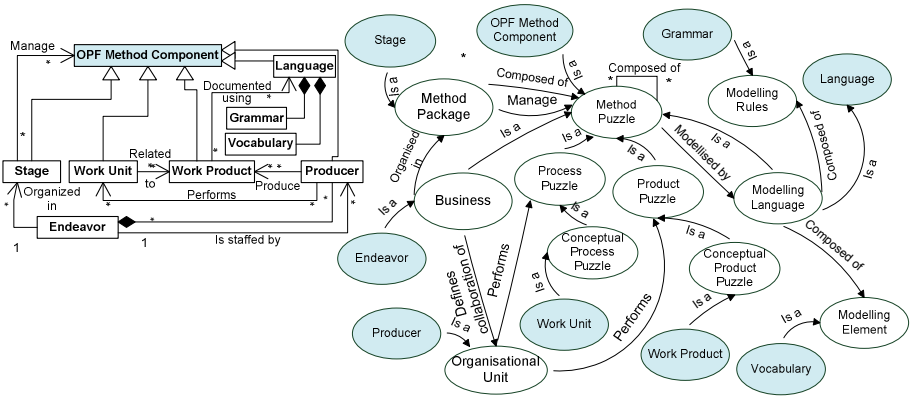


Fig. 5. OPF Method Component Structure

3.5 SO2M Method Services

Introduced by Guzelian and colleagues [10] the SO2M approach is the first step of applying the service oriented paradigm [25] to ME approaches.

A SO2M method service is a reusable unit that contains one or several method fragment to solve an information system development problem [10]. It can be mapped with the *method puzzle* of the ontology and exploit the inclusive property of the *method puzzle* specializations possibilities. A method service is constituted of three abstract parts: identification part, process part and resource part. As these parts are just abstract containers they won't be match with the ontology.

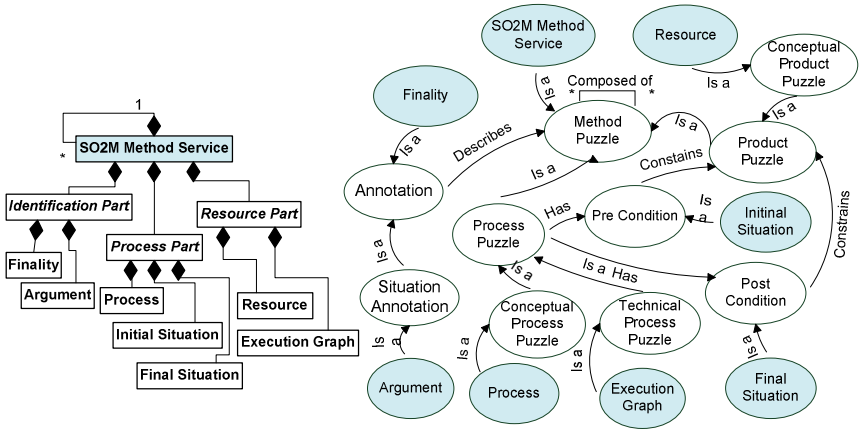


Fig. 6. SO2M Method Service Structure

The identification part aims to capture the contextual knowledge of the method service reuse by defining its finality and argument. A finality is the description of the problem solved by a method service, it's structured with a goal, a manner and a context [10]. As it contains both situational and structural information the finality concept can be defined as a specialization of *annotation concept* in our ontology. The argument concept of the SO2M approach characterizes a method service reuse situation by a list of pro arguments (i.e. advantages) and con arguments (i.e. drawbacks) and it can be matched with the ontology *situation annotation* concept.

The process part is composed of the process initial situation, process final situation and process structure description. These three concepts are respectively matched in our ontology with the *precondition*, *postcondition* and *conceptual process puzzle* concepts.

The resource part defines the implementation of a process by an execution graph which can be mapped as a specialization of our *technical process puzzle* concept. This part also defines the descriptions of all resources consumed or delivered by the process. This latter concept can be defined as a specialization of the *conceptual product puzzle* in our ontology.

This section shows that each concept of the method descriptor ontology is matched in the set of concepts issued of the five studied SME approaches and acknowledges the relevance of the ontology. Furthermore, the matching between the studied approaches and the ontology of method descriptors shows that each of these approaches shares common concepts with the others and also incorporates new concepts to characterize method constructs not addressed in the others. The ontology represents a semantic common ground useful to understand the semantic difference between the various SME approaches. To emphasize the benefit of this ontology and in particular in reducing the ME drawbacks mentioned earlier, three exploring usage scenarios are sketched in the next section.

4 Exploring Usages of the Method Descriptors Ontology

The ontology of method descriptors is an attempt to reach a semantic common ground in SME. This section explores possible usage scenarios of such ontology. Four usage scenarios have been envisioned and described to illustrate the relevance of the ontology-based approach and its usefulness for future ME perspectives.

1. The ontology can be used in an educational manner by non ME experts to understand (i) the basic semantic common ground of the domain and (ii) the various competing modular constructs proposed in SME. This basic usage helps directly in reducing the first ME drawback (confusion).
2. The ontology can be used as a basis of ME reasoning systems such as decisional support system helping ME engineers in their tasks. This usage is complementary to the educational usage.
3. In addition, the ontology can a first step of a process building a unified ME query facility on top of unified Method knowledge Base. Such ME tool is helpful to ME engineers to extract ME knowledge according to their needs expressed in a common language (ontology concepts). Then, a mapping facility must be built to translate the initial query into a specific query compliant to the ME descriptor of the method base. The ontology allows building a generic tool to query method base storing method puzzles belonging to several SME approaches or to query various method bases compliant to SME approaches. Finally, it is a way of reducing ME drawback like lack of SME standard. In fact, the main interest is not in the language used to describe the method puzzle but the fact that the method puzzle matches the ME engineers needs. The perspective of this usage is to propose to the ME community to build a common Method knowledge base which can become a reference for the community and the practitioners.
4. Service orientation in Information Systems leads to re-organize a portfolio of legacy applications into services. By analogy to Information System engineering, we can assume that a service orientation in the ME, leads to re-organize CASE tool into method services (end-user software service). To adapt services oriented technologies to method services, we should extend the service descriptor as it is sketches in Fig. 7.

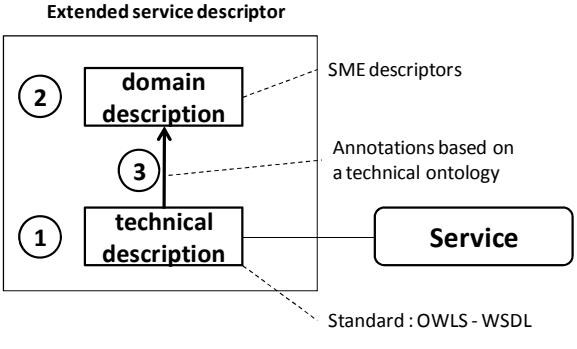


Fig. 7. Principe of service description extension

Figure 7 summarizes YASSA approach [26] to extend the service description to domain description with the usage of a technical ontology. The service description is structured in two layers: technical service description based on standards such as OWLS or WSDL and a domain description represented in our case by any SME descriptor encoded in an XML format. The technical service description refers the domain descriptor through annotations embedded in the technical layer and referring domain value mentioned in the domain description. Annotations are expressed according to the ontology concepts in order to build search algorithms according to the ME ontology instead of the SME descriptor. It means that a Method Service registry is built upon the semantic common ground in SME instead a specific SME approach. This usage allows encapsulation of a tool support part to any SME descriptor with searching algorithms ME ontology compliant but SME descriptor independent. This usage scenario shows a way of reducing indirectly the third ME drawback (lack of implementation tool).

Assuming that a CASE tool is re-organized as a portfolio of method services, and a CASE tool is considered as configurable tools like ERP and product line, the perspective is to configure the method part and the case tool supporting it, at the same time. By analogy to the Product line, this perspective introduces the concept of a Method Line where method is compliant to Seligman definition and the purpose of the CAME is to configure the method and the CASE tool support at the same time. In this case, the CASE tool can be viewed as an assembly of method services which can be combined into a specific configuration.

This section is illustrating how the method descriptors ontology may be used and how it is possible to reduce directly or indirectly the ME drawbacks. These usage scenarios show that a semantic common ground may be enough to step towards in the ME community.

5 Conclusion

In this paper, we have assumed that the diversity in SME approaches is purposeful and shows the richness of the ME community. It is largely agreed that a common ground is needed to overcome some ME drawbacks such as unnecessary confusion for non ME expert, lack of standard & interoperability and lack of implementation tool, but is also a hot topic between researchers. In addition, SME approaches have not been yet largely used by practitioners, or implemented in CAME environment because of these ME drawbacks. The paper proposed an ontology-based approach in SME to build the ontology of method descriptors as a domain ontology. SME approaches promote different method modular constructs but they have a common understanding of what a method is. We exploited the Seligman definition of a method : way of thinking (paradigm), way of working (process), way of modelling (product), way of controlling (organisation) and way of supporting (tool support). Therefore, the ontology defines the core concepts of a method description and the granularity levels built upon them. We assumed that this ontology constitutes a semantic common ground in SME which is a start-up phase in reducing indirectly the ME drawbacks.

However, to be effective, the SME approaches must define their semantic according to the ontology. We showed in this paper how the ontology can be used to define

the semantic of the six most cited SME approaches : 'Method Fragments', 'Method chunks', 'Method components', 'OPF method elements' and 'SO2M method services'. Then, the ontology of method descriptors obtained showed that SME approaches shared common concepts but also incorporated new concepts to characterize methods constructs not addressed in the others. It is why we assumed that differences between SME approaches are purposeful and we have adopted an alternative solution: semantic common ground.

Finally, the paper explored three usage scenarios of the ontology of method descriptors. The ontology can be used as an educational tool for non ME expert to reduce their confusion or as a basis of reasoning systems.
A step forward, the ontology can be used to build a unified ME query facility. In fact, the ontology is used as a mapping tool between the ME engineers query and the technical query executed on a specific method base compliant to a specific SME approach. The benefit of this usage can be to develop one multi-approach method knowledge base which can be the reference of the ME community and can be shared with practitioners.

Moving SME approaches to service orientation, implies to move the CASE tool in the centre of a method description. We illustrated in the paper the application of YASSA's approach to extend service to method service and service description to method service description. A method service description is composed of two related layers: technical and domain service description (ME descriptor) layers. The ontology of method descriptors is used to integrate ME annotations inside the technical service description conform to standards like OWLS or WSDL and it allows to provide searching algorithms of method service built upon the Ontology of Method descriptors instead of the SME descriptor itself. The method service can be described at the domain layer by any SME descriptor.

Finally, the service orientation combines with the ERP or Product line analogy, we can envision the CASE tool and its method description as a method line and its objective is to provide CAME to configure the method description part and the CASE tool at the same time. The perspective is a subject of research.

References

1. Agerfalk, P., Brinkkemper, S., Gonzales-Perez, C., Henderson-Sellers, B., Karlsson, F., Kelly, S., Ralyté, J.: Modularization Constructs in Method Engineering: Towards Common Ground? In: Panel of ME 2007. Springer, Geneva (2007)
2. Deneckère, R., Iacovelli, A., Kornyshova, E., Souveyet, C.: From Method Fragments to Method Services. In: EMMSAD Workshop of CAISE 2008, Montpellier, France (2008)
3. Nehan, Y.-R., Deneckère, R.: Component-based Situational Methods: A framework for understanding SME. In: Situational Method Engineering: Fundamentals and Experiences, Switzerland. IFIP, vol. 244 (2007)
4. Mirbel, I.: Connecting Method Engineering Knowedge: a Community Based Approach. In: Proceedings of ME 2007, Geneva, Switzerland (2007)
5. Niknafs, A., Asadi, M., Abolhassani, H.: Ontology-Based Method Engineering. International Journal of Computer Science and Network Security. IJCSNS 7(8) (2007)

6. Ralyté, J., Deneckère, R., Rolland, C.: Towards a Generic Model for Situational Method Engineering. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681. Springer, Heidelberg (2003)
7. Brinkkemper, S.: Method Engineering: engineering of information systems development method and tools. Information and Software Technology 38(7) (1996)
8. Wistrand, K., Karlsson, F.: Method components – rationale revealed. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 189–201. Springer, Heidelberg (2004)
9. Henderson-Sellers, B.: Process meta-modelling and process construction: examples using the OPF. Ann. Software Engineering 14, 1–4 (2002)
10. Guzélian, G., Cauvet, C.: SO2M: Towards a Service-Oriented Approach for Method Engineering. In: Proceedings of the International Conference IKE 2007, USA, (2007)
11. Rolland, C., Plihon, V., Ralyté, J.: Specifying the reuse context of scenario method chunks. In: Pernici, B., Thanos, C. (eds.) CAiSE 1998. LNCS, vol. 1413, p. 191. Springer, Heidelberg (1998)
12. Seligmann, P.S., Wijers, G.M., Sol, H.G.: Analysing the structure of IS methodologies, an alternative approach. In: Proceedings of the 1st Dutch Conference on Information Systems, Amersfoort, The Netherlands (1989)
13. Plihon, V., Rolland, C.: Modelling Ways–of–Working. In: Iivari, J., Rossi, M., Lyytinen, K. (eds.) CAiSE 1995. LNCS, vol. 932, Springer, Heidelberg (1995)
14. Harmsen, A.F.: SituationalMethod Engineering. Moret Ernst & Young (1997)
15. Rolland, C.: in French: L'ingénierie des méthodes: une visite guidée. In: e-TI (2005)
16. Karlsson, F., Agerfalk, P.J.: Method configuration: adapting to situational characteristics while creating reusable assets. Information Software and Technology 46 (2004)
17. Gonzalez-Perez, C.: Supporting Situational Method Engineering with ISO/IEC 24744 and the Work Product Pool Approach. In: IFIP, Situational Method Engineering: Fundamentals and Experiences (2007)
18. Jackson, M.: Software Requirements & Specifications – a Lexicon of Practice, Principles and Prejudices. ACM Press, Addison-Wesley (1995)
19. Fillmore, C.J.: The case of case. In: Universals in linguistic theory. Holt, Rinehart and Winston Inc. (1968)
20. Dik, S.C.: The theory of functional grammar. Foris Publications, The Netherlands (1989)
21. Open Process Framework, `http://www.opfro.org/`
22. Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-Modelling Based Assembly Techniques for Situational Method Engineering. Information Systems 24(3) (1999)
23. Rolland, C., Prakash, N.: Aproposal for context-specific method engineering. In: Principles of Method Construction and Tool Support, vol. 191-208. Chapman & Hall, Boca Raton (1996)
24. International Organization for Standardization: ISO/IEC 24744, Software Engineering – Metamodel for Development Methodologies
25. Papazoglou, M.P.: Service-Oriented Computing: Concepts, Characteristics and Directions. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering WISE 2003 (2003)
26. Chabeb, Y., Tata, S.: Yet Another Semantic Annotation For WSDL. In: Proceeding of International Conference WWW/Internet IADIS 2008 (2008)

Towards a Method for Service Design

Olga Levina, Trung Nguyen Thanh, Oliver Holschke, and Jannis Rake-Revelant

Berlin Institute of Technology Franklinstr. 28/29, 10587 Berlin, Germany
{olga.levina,oliver.holschke,jannis.rake}@sysedv.tu-berlin.de

Abstract. Services are the vital part of a service-oriented architecture. Their development and design are essential parts of the development and implementation of a service-oriented architecture. Thus, numerous approaches in research and practice exist that refer to different aspects of service design. These are focused on specific needs or aspects in service design. According to the literature review provided in this paper, no single service design approach covers all the aspects that are needed for the implementation and deployment of a service-oriented architecture. Beside the literature review this paper provides a service design approach that combines the existing methods and approaches. The goal is its further development towards a service design method for service design in research and industry.

Keywords: method design, service design, literature review, service-orientation.

1 Introduction

Research activities and practice implementation attempts in the area of Service-Oriented Architecture (SOA) have gained a lot of momentum in the resent years. Methods for SOA development and evaluation started to emerge. Although, despite the evident progress there is still a lack of widely spread methods that can be used in every stage of SOA development and implementation.

SOA is defined here according to OASIS as "a software architecture of services, policies, practices and frameworks in which components can be reused and repurposed rapidly in order to achieve shared and new functionality"[1]. This definition includes one of the main characteristics of SOA: the reuse of the software components, i.e. services. A service can be further described as an element that encapsulates a business function and cannot be further decomposed without harming its functionality. Services can be defined as autonomous, platform-independent entities that can be described, published, discovered and assembled; they are technologically neutral, loosely coupled and support local transparency encapsulating business functionality[2]. Services can be differentiated among others by their goal, functionality, granularity, scope, and interaction.

Service design specifies how the service can be described and therefore found, which business operations underlie the service function, which SOA design principles are supported and which technologies are needed to implement the service. Service design principles are embedded within the general principles of service-orientation that include: Service reuse[3,6,7], formal contract[3,8], lose coupling[2,3], abstraction[3,7,9], composability[3,7], autonomy[3,7], statelessness[3,6], discoverability[10].

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 91–96, 2011.

This paper presents a generalized approach to service design that is based on existing rules for service design and SOA design patterns. The suggested framework aims to support service design by providing a certain procedure and detailed activities in order to enhance the development procedure and to increase the probability for service reuse. The paper is structured as follows: First, the existing service development approaches and methodologies both in research and practice are reviewed. The results of the analysis serve as basis for the development of a canonical framework for service design. Discussion of the results and outlook finish the paper.

2 Existing Approaches to Service Design

Approaches or service design methods reviewed here were chosen using literature research on SOA or service design topic. Their evaluation was not completely based on the method components suggested by [11], because not only service design methods but also approaches explicitly situated in the SOA context were reviewed. Further criteria applied were completeness [11], i.e. the necessary description of the activities, elements and tools as well as realization of the components described in [11], identification process [4], goal reference as well as context of the method (business or software development) [11] and consistency referring to the temporal and logical dependencies between the suggested process steps.

There is little consensus on general service design principles in research and practice. Though, the basic principles for service design can be considered as being: interface orientation, interoperability, modularization and process orientation, e.g. [3, 7-9]. Thus, the service design method or approach needs to include specification on interface design of a service. Besides the technical specification, meta-data on service content and use need to be specified. Communication aspects such as message formats, protocols and addresses need to be included as well as the effect of the service on data, process composition, security and further run-time aspects. Deployment of the interface design needs a service level agreement (SLA) as a realization of the abstraction principle of SOA, as well as its contents need to be included into service design approach. A further advantage lies in the changeability of the software elements.

Interoperability can be defined as the ability of software elements to exchange and interpret information with each other [13, 14]. Modularization implies composition of application systems or business processes into domains or services. Services are supposed to provide flexible support for business process automation. A service design approach or method need to provide guidelines for service granularity of data, business logic as well as functionality.

In the following existing service design approaches or methods are evaluated referring to the above mentioned criteria. The SOA Approach (SOAA)[3] describes top-down and bottom-up service design using examples. It considers aspects of business engineering as well as technical aspects. Design and development of data models and interfaces are described as well as service design pattern are suggested. The service design takes all the basic SOA design principles such as autonomy, lose coupling, statelessness, etc. into account. SOAA also uses industry standards such as XMPL, SOAP, BPEL and UDDI [15, 16].

The Service-Oriented Design and Development Methodology (SODDM)[5] is based on the RUP-Method and analyzes the business requirements for service design. SODDM is based of Web Service[17] technology and does recommend specific vendor tools for service realization. Though the approach is described as a methodology it does not offer a role specification for the design process. The aspect of service orchestration is not further elaborated and service design is concentrated on a fine granular level. The SOA Method (SOAM)[4] puts the focus on the business requirements and business-oriented service realization. Bottom- up as well as top-down service design patterns are conceptualized independently from their domain. SOAM considers the main SOA construction principles, provides a domain model and service classification. This aspect supports a better service reuse and flexible service granularity. Service and interface implementation are mentioned referring to standards such as ebXML, UN/CEFACT[18], etc. for reducing the number of data attributes.

Method For Component-Based And Service-Oriented Systems Engineering (CBSOSE)[20] puts the focus on service identification according to business requirements, business engineering as well as software engineering. In software engineering the focus lies on component-based software development. CBSOSE offers an exhaustive documentation containing action description supported by examples and expected results. Services are modeled using Unified Modeling Language (UML), service granularity is based on business activities. Service reuse is not explicitly considered. The Process Model for Business Service Development (PMBSD) [21] identifies business services according to business requirements. PMBSD provides a role model and generic description of the process activities and results. The SOA design principles are not explicitly mentioned but SLA, reuse and discoverability of services are considered.

Web Service Implementation Methodology (WSIM)[22] focuses on SOA development using Web Service technology. Business engineering is not considered in the methodology though a role model is given. There are any tools specified that can support the methodology in service design. The design principles are not mentioned explicitly but reuse, interoperability and service orchestration are considered. Every phase of the method is supported by best practices. Executive's Guide to Service-Oriented Architecture (EGSOA)[7] approach includes bottom- up and top-down strategy for service identification using business services as well as legacy systems. The focus is on the business aspects of service development. There is no role model present in the method and no examples are provided. The analysis and design phase support the design principles such as SLA, interoperability, reuse, lose coupling and discoverability of the services.

3 Framework for Service Design – Towards a Service Design Method

The suggested framework for service design is based on the SOAM and SOAA approaches described above. SOAM provides a clear process for service identification and system analysis on the business side, while SOAA provides comprehensive description of technical service design.

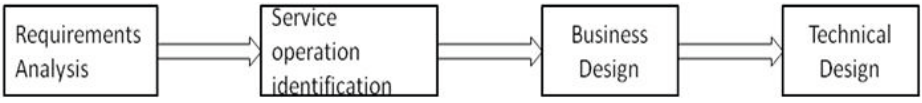


Fig. 1. Basic steps of the service design process

The abstract design process can be divided into four main phases: requirements analysis, identification of service operations, business design, and technical design (see figure 1). The approach includes three actors: Process Owner, Business Process Analyst and IT Developer. The process owner is involved into the process operation, business analyst is a member of a (internal) consulting team or a team that is concerned with business process management tools that are used e.g. for process modeling, and workflow systems. IT developer transforms business requirements into software components. The first two phases are conducted by the process owner and business process analyst. The last two phases are completed by business process analysis and IT developer.

The purpose of the approach is to support business process automation, and integration of technical innovations. Thus, the main sources for the requirements analysis are the business process models and descriptions. The identified operations need to be mapped to the service model. This model is transformed into a service description including interface definition in a WSDL-file.

In the requirements analysis phase business requirements that need to be realized using services are elevated. Here the purpose of the service is defined: encapsulation of a legacy system or business process support. Whether it is possible to support a business process in a SOA is evaluated according to the criteria proposed in [9].

Additionally to the evaluation the business processes need to be captured. As documentation method Business Process Modeling Notation (BPMN) is suggested here as being both easily adoptable to business requirements and providing necessary information for technical implementation. Documentation level is defined by the activities that are refined to the level where further division does not provide any sensitive process information. Besides the activities necessary data object types need to be identified and enriched with accordant attributes. For data object type documentation the UML class diagram is suggested. Resulting process models can be reviewed and examined using the ebXML or RosettaNet[23] standards. OASIS and UN/CEFACT [18] provide context categories for relevant attributes. This information can be used to reduce the information amount.

During the legacy systems analysis application systems supporting the business process are identified. Operations, data objects and interfaces are identified and documented. Legacy systems can be encapsulated using the following possibilities [4]: APIs, Web Services, Data bases, Object oriented software classes, etc.

For business design of the services classes of service candidates need to be identified. This differentiation allows the application of service design principles to each service. Here basic, process-oriented, public enterprise and semi-services are taken into account [8]. During the business-oriented service design identified operations need to be analyzed regarding their granularity and grouped to service candidates. Operations can be grouped according to the user rights or to the data object type. Third possibility includes operation grouping according to their task or operational

domain. Documentation of the identified operations and the grouped services can be achieved using a simple table tool.

During the technical service design a technology for service realization needs to be chosen. Here the Web Service technology is suggested due to its popularity and a quasi-standard status in research and practice. Basic Profile including WSDL 1.0 [24], UDDI 2,0, SOAP 1.0, XML 1.0 [25], XML Schema 1.0 can be used for service specification. It can be supported by the use of the WS- extension frameworks. Development patterns that can enable statelessness on the technical level are e.g.: Atomic Service Transaction, State Messaging, etc.

4 Discussion and Outlook

The presented approach provides a standardized procedure to service design deriving its concepts from existing methodologies and technological approaches. The method needs to be enriched with approximate time-levels for each of the phases as well as a detailed result description for the results of each phase. The roles concept cannot be précised any closer as it often depends on the given circumstances in the enterprise. The suggested role concept assumes C-level management support for the SOA implementation as well as know-how in business (process) analysis and documentation methods. Communication and information structures need to be specified in detail to allow management and control of the implementation initiative. Another assumption here is the cooperation of the business and IT departments that is facilitated by a mediation team of interdisciplinary analysts.

The presented approach can be evaluated using the criteria for method description [11] with the result, that the approach cannot yet be referred to as a method, as the constructs used and principles of form and function are not elaborated extensively. Testable propositions also need to be defined and applied in a subsequent case study.

Hence, to be able to be referred to as a method the approach has to provide some considerable aspects such as construct definition and description, the review of principles of form and function as well as testable propositions. These aspects will be deepened in the future work. The founding will be applied on several case studies so that the method can be further refined and made more feasible for practitioners and researchers.

References

1. OASIS. Reference Model for Service Oriented Architecture 1.0 2006 (2006), http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf
2. Papazoglou, M.P.: Service -Oriented Computing: Concepts, Characteristics and Directions. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering (2003)
3. Erl, T.: Service-Oriented Architecture: Concepts Technology, and Design, 5th edn. Prentice Hall PTR, New Jersey (2005)
4. Offermann, P., Bub, U.: A Method for Information Systems Development According to SOA. In: AMCIS 2009 (2009)

5. Papazoglu, M., van de Heuvel, W.-J.: Service-Oriented Design and Development Methodology. International Journal of Web Engineering and Technology, 412–442 (2006)
6. Josuttis, M.N.: SOA in Practice, 1st edn. O'Reilly, Sebastopol (2007)
7. Marks, E.A., Bell, M.: Service-Oriented Architecture: A Planning and Implementation Guide for Business and Technology, 1st edn. John Wiley and Sons Inc., Hoboken (2006)
8. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Roads and Best Practices for Service-Oriented Architectures, 1st edn. REDLINE GmbH, Heidelberg (2007)
9. Heutschi, R.: Serviceorientierte Architektur 2007. Springer, Berlin (2007)
10. Fringer, P., Zeppenfeld, K.: SOA and Web Services, 1st edn. Springer, Heidelberg (2009) (in German)
11. Offermann, P., Blom, S., Levina, O., Bub, U.: Proposal for Components of Method Design Theories. Wirtschaftsinformatik/Business & Information Systems Engineering 52/2(5/5), 287–297, 295–304 (2010)
12. Beverungen, D., Knackstedt, R., Müller, O.: Development of Service-Oriented Architectures for Manufacturing and Service Integration (2008) (in German)
13. Thomas, O., Leyking, K., Scheid, M.: Service-, Prozess Models for Service-Oriented Software Development. In: Wirtschaftsinformatik 2009, pp. 181–190 (2009) (in German)
14. OASIS. OASIS Standard 20041: Web Services Security: SOAP Message Security 1.0, WS-Security 2004 (2004)
15. OASIS. Web Services Business Process Execution Language Version 2.0 OASIS Standard 11 (2007) (cited)
16. W3C. Web Services Architecture (2004)
17. Huemer, C. (ed.): UN/CEFACT, UN/CEFACT's Modeling Methodology (UMM) UMM Meta Model – Base Module Candidate for 2.0 Public Draft (2008)
18. Mayerl, C., Link, S., Racke, M., Popescu, S., Vogel, T., Mehl, O., Abeck, S.: Method for Design of SLA-enbaled IT-Services. In: Müller, P.G., Gotzhein, R., Schmitt, J.B. (eds.) Communication in Distributed Systems. Springer, Heidelberg (2005)
19. Stojanovic, Z.: A Method for Component-Based and Service-Oriented Systems Engineering (2005)
20. Stein, S., Ivanov, K.: Process Model for Business Service Development (2006)
21. Lee, E.W., Haines, M., Chan, L.P., Ang, C.H., Tan, S.P., Lee, H.B., Cheng, Y., Xu, X., Yin, Z.: Web Services Implementation Methodology (2005), `http://www.oasis-open.org/committees/download.php/21354/fwsi-im-1.0-guidelines-doc-wd-PublicReviewDraft1.0.pdf`
22. RosettaNet (2010), `http://www.rosettanet.org/` (cited)
23. W3C. Web Services Description Language (WSDL) 1.1 (2001)
24. W3C. Extensible Markup Language, XML (2003)

A Case Study for Improving a Collaborative Design Process

Sophie Dupuy-Chessa, Nadine Mandran, Guillaume Godet-Bar, and Dominique Rieu

University of Grenoble, CNRS, LIG
385 rue de la Bibliothèque
38041 Grenoble Cedex 9, France
Firstname.Lastname@imag.fr

Abstract. We propose a design method for supporting the design of rich user interfaces. It integrates software engineering and human-computer interaction practices through collaborations and focuses on the traceability of processes and models. In this paper, we investigate these collaborative aspects with a case study, which gave us some insights in order to improve the process.

Keywords: Collaboration, Process, Qualitative study.

1 Introduction

The Software Engineering (SE) methods have shown their reliability for specifying and developing the functional core of information systems. Nowadays, such systems can have rich user interfaces based on interaction techniques like vocal commands or gesture recognition. To guide their design, we propose the Extended Symphony method [1]. It has been designed for facilitating collaborations between SE and Human Computer Interaction (HCI) specialists and for enabling designers to develop rich user interfaces. But it was still a theoretical proposal that needed to be confronted to practical issues. Then we realized empirical studies focused on specific parts of the method [2]. One of them is presented in this article. It gave us some insights about the collaborative aspects of the method. It studied two hypotheses made while designing the extension of Symphony: 1) the process facilitates the collaboration between actors from the SE and HCI domains and 2) it allows designers to produce consistent models.

The following section gives an overview of the collaborative process of the Extended Symphony method. Then we present the case study, before concluding with some perspectives.

2 The Extended Symphony Method

Originally developed by the UMANIS Company, Symphony is based on a Y-shaped development cycle whose functional (left) branch corresponds to the traditional task of domain modelling, independently from technical aspects. This branch whose an excerpt is given in Fig. 1, focuses on the integration of SE and HCI practices.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 97–101, 2011.

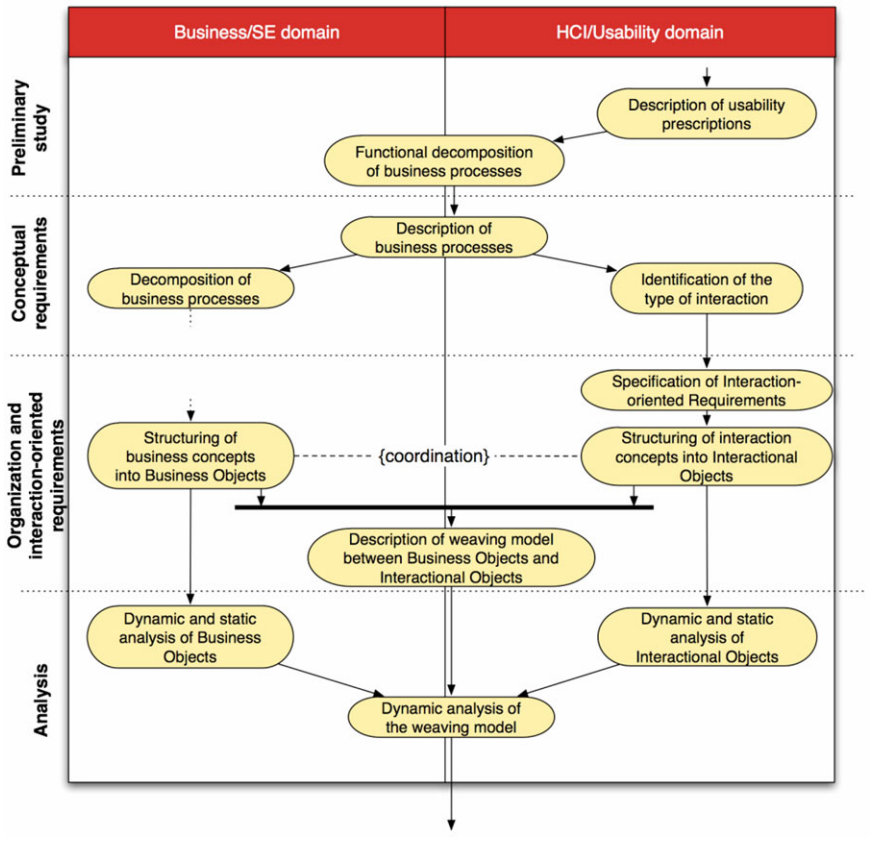


Fig. 1. Collaborations during the functional branch

The process starts with a cooperation study phase whose goal is providing a common decomposition of business processes. Then for each business process scenarii are described to start with a common view of the application. Then each specialist works in parallel with her own models: for HCI, task trees [3] and interaction model [4] for rich user interfaces; use cases, sequences diagrams for SE. From these models, the SE and HCI specialists must structure their concepts using Symphony Objects that are reusable components. The interaction space is structured with Interactional Objects (IO [5]), which are user interface-oriented components. The business is designed into Business Objects (BO).

Then the SE and HCI experts identify whether they need to modify their models to facilitate their ulterior weaving. It is a coordination activity: the experts do not need to produce a common product; they compare their models to validate their design choices.

The organizational and interaction-oriented requirements phase ends with a cooperation where the design actors must work together to produce a common product. The "Description of weaving model between BO and IO" allows both the HCI and SE experts identifying which IO correspond to projections of BO.

Finally the functional branch terminates by an analysis phase where the models are completed by refining the weaving model. It is not studied here, as it is only a refinement of the previous phases.

3 Case Study

3.1 Setting of the Case Study

Rather than considering the process performance, we focus on "what are the right things to do" for designing a system with a rich user interface. We choose a qualitative approach to gather an in-depth understanding of the subject studied, with smaller but focused samples. With this approach, a comparison with another method mixing HCI and SE practices [6,7] would be useful; but it would be difficult to realize as many variables need to be controlled to obtain a useful experiment.

The experimental design is inspired from the social probes [8], translated to the professional context. It is based on the use of treatment groups only (no control group) and on a qualitative collection of data.

Participants. Four groups of two designers were asked to specify the same system. They are members of research groups specialized in HCI and SE, with more or less the same profile. They are colleagues of the experimenters. None of them was introduced to the Extended Symphony Method beforehand and they were volunteers to use it.

Steps. All the groups worked on the same example (a collaborative tool for designing public spaces). They followed mainly five steps during one week: 1) The first step is the starting session: a questionnaire about work habits, a short introduction to the Extended Symphony Method and the example were given to the participants. 2) Each group had to work on the example. Participants worked along or with the specialist of the other domain. Each time that a designer worked, he recorded his results, the time spent, his goals, his difficulties in a form, giving us so a probe. 3) After four days, each group had to realize the cooperative activity “Description of weaving model between BO and IO” (Fig. 1) to link the HCI and SE models. This intermediary session was recorded to evaluate whether the HCI and SE models were difficult to merge. 4) The groups had other working sessions to finalize their proposal. 5) They presented their results and during a focus group session, they gave their opinion on the process. One of the participants was absent at this last session.

3.2 Results

The method was perceived as interesting and satisfying. Collaboration was mainly perceived as useful (5/7). It was cited as one of the elements to reduce errors thanks to a better understanding between people of different domains. One positive aspect for collaboration is the separation of concerns between SE and HCI. But the participants also appreciated the common vision, facilitated by the use of common models. Nevertheless one of HCI participants pointed out the necessity of a common approach between the two specialists.

Globally the sequencing of activities was considered logical and natural. However three participants thought that the process can be long even if for the majority of the participants (6/7), it can make the system design more efficient. The main reasons given by the participants were: 1) the collaborators start the design with a shared vision (the initial scenario); 2) The designers must think at the appropriate level of abstraction according to the design process.

Regarding the efficiency, the duration of the project and of the collaborative exchanges varied a lot from one group to another. Two groups spent less than 4 hours of work while another one took four times more. One group spent less than 15% of its total work time on collaborative activities, while another group spent 85% of its total work time. In the second group, each specialist did not respect his role and most of the work was realized in cooperation.

From the model viewpoint, it has been pointed out that the process gave rise to too many models. It can become difficult to check their consistency. However in all the groups, scenarii were a reference for model consistency. Its use was perceived as facilitating cooperation at the beginning of the process. For the other models, amongst the four groups, two have not realized the weaving between Symphony Objects. But the HCI and the SE models were judged as consistent because the common concepts of the two domains were identified and named in the same way. For the groups that used Symphony Objects, we noted that some adjustments (addition/suppression of objects) were made to obtain consistent models during the intermediary session. The specialists (3/4) considered that Symphony Objects are "a bridge between SE and HCI, a good synchronization point".

4 Evolution of the Extended Symphony Method

As we mentioned previously, some drifts in the process were noticed: 1) the process can be too long. 2) Some collaborative activities can occur for inappropriate goals although they are not in the process.

The participants suggested us the following improvements: 3) the steps where the consistency between SE and HCI models must be checked must be more explicit. 4) The method should be adapted to the project size. 5) A glossary could be added to provide a clear and short description of concepts. 6) A description of the role of each specialist could help each one in understanding his role. 7) A more precise description of Symphony Objects could be provided.

The first evolution answers to the points 1 and 4. The process was simplified. Many activities became optional. Only the activities that produce the essential models remain mandatory. These models are those used to communicate with the stakeholders (e.g. scenarii), or to concretize the collaboration (e.g. Symphony Objects model).

For points 2 and 3, we considered that our two types of collaborations were disturbing. There was a misunderstanding about the coordination notion. Now we only propose cooperative activities whose goal is clearer. The description of collaborative activities has also been enriched: each activity is now considered from the viewpoint of the responsibilities of each actor. This is also a partial answer to point 5.

Globally we improved the documentation to respond to the three last points: a definition of a term is given when using it (point 5); we added a description of each role at the beginning of the method description (point 6); we tried to be more precise about the level of abstraction expected for each model (point 7). We systematically introduced examples of the expected products in the description of an activity.

5 Conclusion

This paper describes a case study that gave us some insights about the Extended Symphony collaborative process. Even if this case study has no statistical value, it was interesting in a qualitative approach to gather a variety of feedbacks. It allowed us improving the process by simplifying it. Of course these improvements would need to be evaluated by some experiments. More generally, we argue for a more systematic use of the qualitative approach for method engineering. With this goal, we are currently working with evaluation specialists to describe some of their knowledge in a reusable manner.

References

1. Dupuy-Chessa, S., Godet-Bar, G., Pérez-Medina, J.-L., Rieu, D., Juras, D.: A Software Engineering Method for the Design of Mixed Reality Systems. In: Engineering of Mixed Reality, ch. 15. Springer, Heidelberg (2009)
2. Ceret, E., Dupuy-Chessa, S., Godet-Bar, G.: Using Software Metrics in the Evaluation of a Conceptual Component. In: 4th Int. Conf. On Research Challenge in Information Science RCIS 2010, Nice, France (2010)
3. Paterno, F.: ConcurTaskTrees: An Engineered Notation for Task Models. In: The Handbook of Task Analysis for Human-Computer Interaction, pp. 483–503. Lawrence Erlbaum Associates, Mahwah (2003)
4. Dubois, E., Gray, P., Nigay, L.: ASUR++: a Design Notation for Mobile Mixed Systems. Interacting With Computers 15(4), 497–520 (2003)
5. Godet-Bar, G., Rieu, D., Dupuy-Chessa, S., Juras, D.: Interactional Objects: HCI concerns in the analysis phase of the Symphony method. In: 9th International Conference on Enterprise Information Systems ICEIS 2007, Funchal, Madeira, pp. 37–44 (2007)
6. Fox, D., Sillito, J., Maurer, F.: Agile methods and User-Centered Design: How these two methodologies are being successfully integrated in industry. In: Proceedings of Agile 2008, pp. 63–72. IEEE Computer Society, Washington, DC, USA (2008)
7. Corlett, D.: Design: innovating with OVID. Interactions 7(4), 19–26 (2000)
8. Bernhaupt, R.: Usability and User Experience Evaluation in Non-Traditional Environments, HDR de l'Université Paul Sabatier, Toulouse (2009)

Incorporating Model-Driven Techniques into Requirements Engineering for the Service-Oriented Development Process

Grzegorz Loniewski, Ausias Armesto, and Emilio Insfran

ISSI Research Group, Department of Computer Science and Computation
Universidad Politécnica de Valencia
Camino de Vera, s/n, 46022, Valencia, Spain
grlo@posgrado.upv.es, aarmesto@indra.es, einsfran@dsic.upv.es

Abstract. Modern information systems, which are the result of the interconnection of systems of many organizations, run in variable contexts, and require both a lightweight approach to interoperability and the capability to actively react to changing requirements and failures. *Model-Driven Development* (MDD) and *Service-Oriented Architecture* (SOA) are software development approaches that deal with this complexity, reducing time and cost development and augmenting flexibility and interoperability. Although, requirements engineering is accepted as a critical activity in these approaches, there is a need to appropriately integrate and automate the requirements modeling and transformation tasks as part of MDD and SOA development approaches. Our proposal is a *Rational Unified Process* (RUP) extension, in which the requirements discipline is placed in a model-driven context in order to derive SOAs. This paper includes the definition of a model-driven requirements process including activities, roles, and work products.

Keywords: Model-driven development, SOA, RUP extension, Requirements Engineering.

1 Introduction

The domains and problems for which it would be desirable to introduce information systems are currently very complex and the software development process is thus of the same complexity. Several software development approaches have been introduced in order to speed up and facilitate this process through its automation and the division of the final product into smaller building blocks.

One of these approaches is Service-Oriented Architecture (SOA). SOA is a logical means of designing a software system to provide independent services that are aligned with business processes. SOA strengthens such factors as reusability, scalability or interoperability.

Another approach that improves the development process of complex applications is Model-Driven Development (MDD). This is a model-based approach that promotes the separation of concerns between the business specifications and

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 102–107, 2011.

their implementation. This separation is achieved through the use of models that allow the level of abstraction to be elevated. It provides a means for development process automation by model transformations and code generation rules.

These approaches are very often used during the design phase of software development, less often in the analysis phase, and hardly ever in the initial phase of a software project when requirements have to be captured, understood and specified. Moreover, even though the aforementioned approaches provide the means to support the software development process, all such techniques, methods or architecture styles are of little use without a well-defined process that places them in a particular context.

In our opinion, the solution to providing a successful automatable development of SOA-based systems is a well-defined, and flexible model-driven process, which is requirements engineering (RE). A good basis for the development of such a methodological approach is the Rational Unified Process. RUP is a costumizable and extensible software engineering process that provides a disciplined approach with which to define tasks and responsibilities in an organized system development [5]. Although various attempts to adapt RUP to MDD principles exist, e.g. Agile Unified Process (AUP), the development process remains mainly manual.

This paper presents a proposal for a RUP extension and adaptation with which to develop SOA-based systems by using model-driven techniques. The main extension in this methodology is the replacement of the Requirements discipline with the Model-Driven Requirements. This work can be considered as an interesting contribution for those software process engineers who are faced with the challenge of guiding software development projects that follow a model-driven development approach from the requirements elicitation.

This work is structured as follows. Section 2 presents works related to the aforementioned area of concern. Section 3 provides an overview of the software process engineering standards. Section 4 presents an overview of the main goals of the methodology, focusing on the content and process elements of the Model-Driven Requirements discipline in the context of SOA-based systems development. Finally, Section 5 contains some conclusions and future work.

2 Related Works

A variety of modeling techniques and methodological approaches for service-oriented software development have been published in literature. Ramollari *et al.* [9] present a state-of-the-art survey on current service-oriented development approaches, among others, *Service Oriented Unified Process* (SOUP) [7] and *Service Oriented Modeling and Architecture* (SOMA) [1]. However, none of these methodologies describes a complete methodological automated process that includes RE techniques.

There exist other approaches not included in the aforementioned survey, such as: MINERVA framework [3], or SOA-MDK [2], which apply model-based paradigms to service-oriented development methodology. However, these

approaches do not include any automation while producing the services specification. SOA-MDK approach proposes the application of the Model-Driven Architecture (MDA) principles within the context of reference models. However, the nature of the model-driven base of this approach remains unclear.

Several generic methodologies are based on the MDD principles, since these have gained many enthusiasts over the last decade. However, to the best of our knowledge, a complete development process for MDD that incorporates the requirements techniques has not been defined [6]. One such approach is OpenUP/MDD, which is a very simplified RUP version intended for small teams. It is consistent with the MDA, but focuses solely on the transformations from the PIM to the PSM level and does not cover transformations from requirements. In this context, our proposal for the RUP extension and the OpenUP/MDD approach are complementary.

3 Software Process Engineering

Different software development processes use different concepts and notations to define the contents of the methodology. The need to unify all these concepts and notations has therefore emerged leading to the introduction of the *Software Process Engineering Metamodel* (SPEM) [8] standard by the OMG. SPEM provides a complete metamodel based on the *Meta Object Facility* (MOF) to formally express and maintain development method content and processes. The *Unified Method Architecture* (UMA) is an evolution of SPEM v1.1 and defines the schema and terminology used to represent methods consisting of method content and processes. IBM and OMG have worked on UMA to make it part of SPEM 2.0. The UMA engineering process is employed in this extension, defined by the use of IBM Rational Method Composer (RMC) [4], which is a UMA-based comprehensive process authoring tool that provides extensive method authoring and publishing capabilities [10].

4 RUP Extension for the Model-Driven Requirements

In classic RUP, the *Requirements* discipline serves to establish the agreement with customers with regard to what the system should do, and define boundaries of the system. In our opinion, it should also provide a means for developers to better understand the requirements, it being like a bridge between the domain experts, stakeholders and the IT people.

Figure 1.A illustrates the RUP hump chart in which the *Requirements* discipline is replaced with a new *Model-Driven Requirements* (MDR) discipline. It also emphasizes the *Environment* discipline which serves as a means to adapt this process to SOA-based systems.

As shown in Figure 1.A, the new MDR discipline is a concern from the *Inception* phase to the *Transition*. Since the hump chart emphasizes the workload within disciplines, the diagram shows that the new discipline is particularly important during the *Inception* and *Elaboration* phase, in which the product vision

is created and the architecture is established. Since we concentrate on model use in the MDD context, the workload in the *Analysis & Design* discipline in the *Elaboration* phase decreases depending on the degree of automation of activities from the MDR discipline.

This approach was designed to support SOA-based system development. One of the main differences between RUP and the process proposed in our extension is the approach used to relate requirements and the system architecture. In RUP, the architecture is defined on the basis of previously created use cases and scenarios, chosen as the requirements that define strategic architectural elements. This RUP extension is architecture-oriented. It is the architectural pattern identified for the system, in this case to SOA, that becomes a basis for the MDD process definition.

4.1 Activities and Workflow

A set of new activities is contributed and the discipline workflow has been replaced. Figure 1.B demonstrates the MDR discipline workflow. New or altered activities introduced with regard to the classic RUP *Requirements* discipline are marked with a star. Owing to space constraints, we shall comment only briefly on the newly introduced activities, with which the PIM-level model is defined and generated.

Identify a Candidate Architecture. This activity is performed in the early *Elaboration* phase and is essential activity for the software development process in that it determine which artifacts need to be developed (type of model at the PIM-level that the architecture implies), and the MDD process to be followed.
Define the Transformation Rules. This activity is the most essential in this approach. Within this activity, the elements of the source and target models are identified and well-documented. The transformation language is also chosen, and the transformation automation level and tool support are specified. Transformation rules are described in a specially prepared Transformation Rules Catalog.

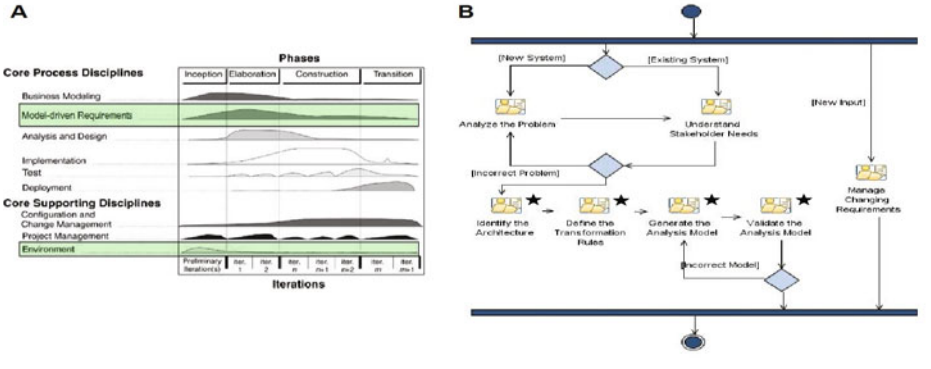


Fig. 1. A) RUP Extension disciplines, B) Model-Driven Requirements Workflow

Generate the Analysis Model. This activity concludes the entire requirements modeling process by creating an architecture-oriented PIM. This is the result of all the previously performed activities, taking advantage of the artifacts created in the *Business Modeling* discipline. The output product of this activity is the input for further process analysis, design and implementation tasks.

4.2 Work Products

Owing to space constraints, we shall comment only briefly on the most important artifacts which have been introduced or improved in the new discipline.

Software Architecture Document (SAD). This artifact, from the *Analysis & Design* discipline, is here initiated on the basis of the system architecture that has been settled on. It is an important artifact for architects and analysts during the entire development process.
Transformation Rules Catalog (TRC). The transformation rules are specified on the basis of the source and target model elements identified. This artifact should consist of a precise description of rules, mappings and refinements, which also provides the basis for the requirements traceability.
Transformation Iteration Plan (TIP). Requirements transformations are usually quite complex and are frequently based on defining intermediate models. A sequence of transformations rather than a single transformation is therefore necessary. This artifact is created to plan a logical order of the transformation to be performed.
Generated Analysis Model (GAM). This is the most important work product in the discipline, it being a source for further transformations to generate PSMs. Its type of content depends on the architecture identified, while the model must suit the architectural pattern considered.

4.3 Roles

As the new discipline is based on the *Requirements* discipline, it maintains the roles originally defined by RUP. The only exception is that the *Requirements Specifier* has been replaced with two additional roles related to the model-driven context activities: *Model Analyst* and *Transformation Specifier*. Only the newly introduced roles are briefly described owing to space limitations.

Model Analyst. During the MDR discipline, the *Model Analyst* coordinates a number of tasks related to: model transformations, model traceability and model validation. The main artifact for which this role is responsible is the GAM. This role also collaborates with the *System Analyst* to accomplish a number of tasks related to requirements modeling and traceability.
Transformations Specifier. This role is responsible for specifying the details of transformation rules to transform requirements model into analysis model. It is a good practice to establish such rules in the meta-model level, which also simplifies the requirements traceability.

5 Conclusions

In this paper we have presented an extension of RUP by placing emphasis on the use of models as requirements representation in the context of MDD. This extension proposes a new discipline called Model-Driven Requirements that substitutes the Requirements discipline from the classic RUP. This approach through the application of architecture-oriented model-driven techniques attempts to extend RUP to specific project needs. It improves the standard development process defined by RUP in that it is not only model-based, but also model-driven.

This extension includes new content elements, such as: artifacts, roles, tasks, activities and capability patterns, to guide software engineers who attempt to follow an MDD approach in their software projects.

As further work, we plan to validate the approach by measuring the effort involved in the maintainability of requirements and the number of failures caused by errors in preparing the requirements specification in comparison to other similar sized projects carried out with the use of classical methodologies.

References

1. Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Gariapathy, S., Holley, K.: SOMA: a method for developing service-oriented solutions. IBM Syst. J. 47(3), 377–396 (2008)
2. Barn, B., Dexter, H., Oussena, S., Sparks, D.: Soa-mdk: Towards a method development kit for service oriented system development. In: Magyar, G., Knapp, G., Wojtkowski, W., Wojtkowski, W.G., Zupancic, J. (eds.) Advances in Information Systems Development, pp. 191–201. Springer US, Heidelberg (2008)
3. Delgado, A., Ruiz, F., de Guzmán, I.G.R., Piattin, M.: A Model-driven and Service-oriented framework for the business process improvement. Journal of Systems Integration 1(3) (2010)
4. Haumer, P.: IBM Rational Method Composer: Part 1: key concepts (December 2005), http://www-128.ibm.com/developerworks/rational/library/jan06/haumer/
5. Kruchten, P.: The Rational Unified Process: an introduction. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)
6. Loniewski, G., Insfrán, E., Abrahão, S.: A systematic review of the use of requirements engineering techniques in model-driven development. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010. LNCS, vol. 6395, pp. 213–227. Springer, Heidelberg (2010)
7. Mittal, K.: Service Oriented Unified Process, SOUP (2006), http://www.kunalmittal.com/html/soup.shtml
8. OMG (Object Management Group): Software Process Engineering Metamodel (SPEM) (April 2008)
9. Ramollari, E., Dranidis, D., Simons, A.J.H.: A survey of service oriented development methodologies, http://staffwww.dcs.shef.ac.uk/people/A.Simons/research/papers/soasurvey.pdf
10. Shuja, A., Krebs, J.: IBM®Rational Unified Process®Reference and Certification Guide: Solution Designer. IBM Press (2007)

The Online Method Engine: From Process Assessment to Method Execution

Kevin Vlaanderen, Inge van de Weerd, and Sjaak Brinkkemper

Utrecht University,
Department of Computer Science,
P.O. Box 80.007,
3508 TA, Utrecht, The Netherlands
{k.vlaanderen,i.vandeweerd,s.brinkkemper}@cs.uu.nl

Abstract. The field of method engineering has seen an increasing amount of interesting approaches and techniques over the last ten years. The coverage of these techniques ranges from the modeling of processes and systems to the situational construction of new ones. However, access to the required domain knowledge is often not available, and the effort required for effective method engineering is in most cases too much. To overcome these problems, we propose an incremental approach for process assessment, process improvement, and process execution, based on method engineering techniques and tools. The approach is implemented in the Online Method Engine; a holistic solution that supports these three aspects. In this paper, we give a conceptual overview of the approach, along with an overview of the current state of development.

Keywords: Method Engineering, Assessment, Process Improvement, Online Method Engine, Method Fragments, Method-as-a-Service, Software Product Management.

1 Introduction

Many researchers [17,15,10,4] describe the use of a method base in situational method engineering. Method fragments can be stored in it, for example by using the MEL method engineering language [6]. Once retrieved from the method base, they can be combined following the assembly rules described by Brinkkemper [5]. More recently, work has been performed on allowing incremental method evolution [23]. According to this work, method fragments can be used to describe and improve the evolution of software product management methods, by allowing the insertion, modification and deletion of method fragment components.

Some method bases have actually been implemented, such as OPF [11] and the CREWS method base [14]. However, for practitioners (the actual method users), retrieving these method fragments and using those in their daily work can be cumbersome. A prerequisite is that the method user should be aware of the exact method fragment that he or she is searching for. In addition, the method user must know what to do with the retrieved method fragment, how to interpret it, and how to implement it in the organization.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 108–122, 2011.

To support the method engineering activity, several computer-aided method engineering (CAME or meta-CASE) tools have been developed. Most of these focus on the meta-modeling aspect. One well-known example of such a tool is MetaEdit+ [19], which enables the definition and usage of domain-specific languages. This tool was also applied in a more agile context [3]. On the other hand, several tools that focus more on the method construction aspect have been developed as well. One example in this field is the work of Saeki [18]. Work on the method base management system 'Decamerone' has been performed by Harmsen and Brinkkemper [9].

Unfortunately, the current method bases and knowledge infrastructures are too hard to use for many practitioners. They do not always know exactly what they are looking for, nor how to implement a formal method description in the processes of their organization. Therefore, in this research, we go a few steps further. We propose an Online Method Engine (OME) that can not only be used to store and retrieve method fragments, but also to assess an organization's current processes, create an advice based on this assessment, and implement this advice in the organizations processes and tools.

This research has many similarities with research on the Method as a Service, described by Rolland [16] and Deneckère et al. [7], and by Guzélian and Cauvet [8]. By adopting a Service-Oriented Architecture (SOA) for method engineering, the authors aim to change method fragments into method services which are implemented as Web services [16]. Deneckère describes how the concept of SOA is adopted in a MOA, a Method-Oriented Architecture. This MOA facilitates a method services registry in which available method services are organized. The authors describe the MOA usage in two use cases. They state that method engineers can use CAME tools to define new method with services compositions. On the other hand, method users (developers, practitioners) can use their CASE-tools to invoke remote method services. Unfortunately, the Method as a Service concept is not thoroughly understood yet.

In our vision of the OME, existing method bases are extended. The OME does not only provide a repository in which method fragments are stored, but also offers the opportunity for users to assess their own processes and investigate which ones should be improved. Based on this assessment, an improvement roadmap is created that is used as a basis for a number of method increments. Furthermore, the company's tooling infrastructure can be directly aligned with the method improvement by automatically configuring templates and work-documents.

In the remainder of this paper, we first explain the principle of model-driven process assessment and improvement. Then, section 3 describes the implementation of this principle in the OME. Finally, in section 4, we present our conclusions and further research.

2 Incremental Process Assessment and Improvement

The idea of incremental process improvement that we present in this paper consists of several separate steps. The starting point for each process improvement

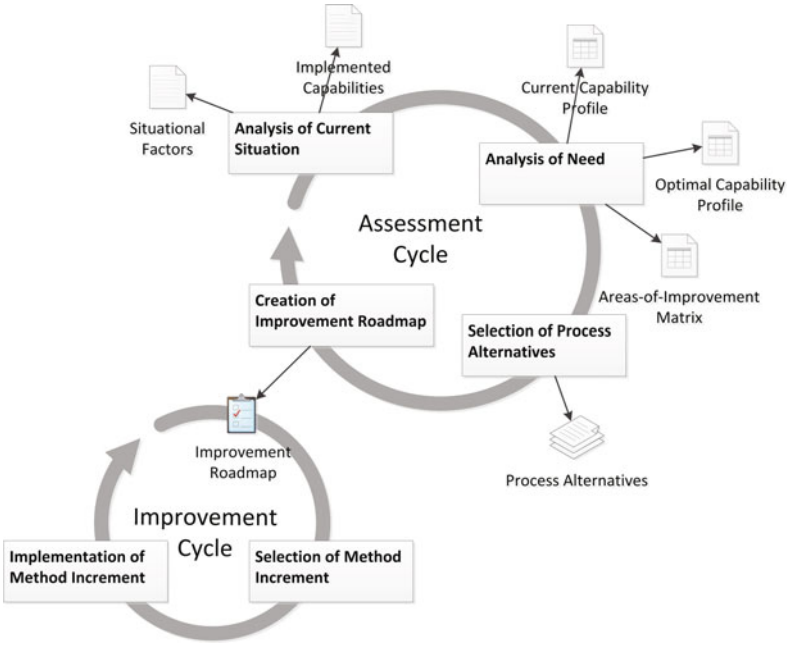


Fig. 1. Incremental Process Improvement

is an analysis of the current process, based on which a maturity profile can be calculated. The situational factors of the company are used to determine an optimal maturity profile. By calculating the delta between these two, the required process improvement is determined. This process improvement is further detailed by relating it to suitable method fragments that can be combined into a new process that improves the company's process. This brief summary of the process is illustrated in Figure 1. Each of the steps in the model is explained in more detail in the following sections.

At several points in the text, references will be made to example implementations in the domain of Software Product Management (SPM). In contrast to most method engineering approaches, our solutions are not implemented in the software engineering domain. SPM deals with management of requirements, the definition of releases, and the definition of software products in a context where many internal and external stakeholders are involved [20]. It represents a context where the creation and application of situational methods is very relevant, but where knowledge regarding effective method implementations is scarce.

2.1 Analysis of Current Situation

The first step in the process improvement activity is obtaining an overview of the current situation in terms of implemented capabilities, and situational factors of the business (unit). This approach can be generalized into a form as depicted

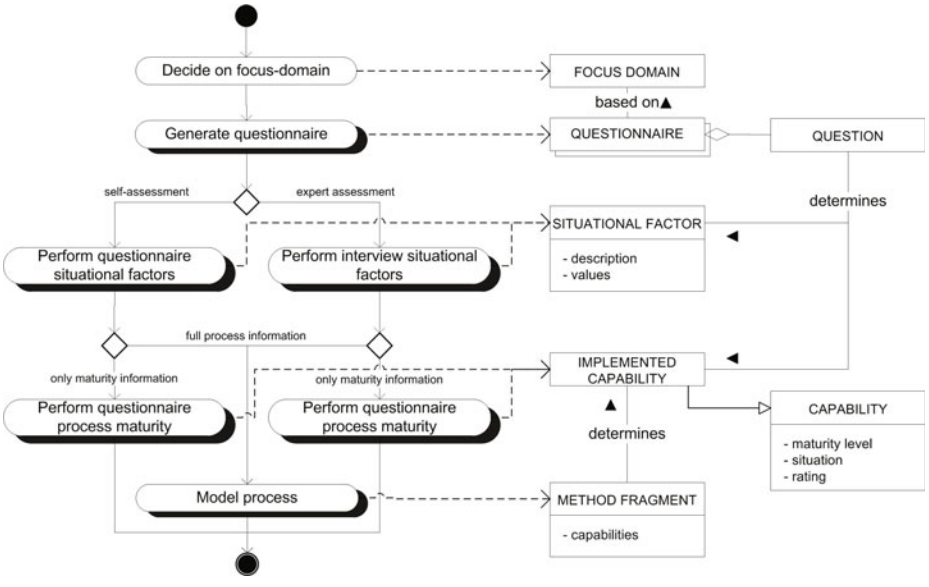


Fig. 2. Analysis of current situation

by Figure 2. The figure is an instance of a PDD (the same applies to Figure 3). Its notation is based on a combination of a UML activity diagram and a UML class diagram. On the left-hand side, boxes indicate the activities that are to be performed. Complex activities that are not further specified here have a black shadow. On the right-hand side, the resulting deliverables are shown, along with their relationships.

The current situation constitutes both the currently employed process as well as more generic aspects of the company at hand. During the initial phase, it is the company that needs to decide what the extent of the analysis will be, i.e. the focus domain. We can identify two types of situations regarding the motivation for employing method engineering:

- The need for improvement of a specific area. In many cases, method improvements can better be performed in an evolutionary way rather than in a revolutionary way. By doing so, you reduce risk and increase the chance of success. This also means that it is often not required to analyze the entire process. Instead, only a specific part of the process is looked at, and only for that part improvements are provided.
- The need for improvement of the entire process. For companies that do require a major improvement of their process, this should be a possibility. In those cases, the entire process should be analyzed. This group also contains (new) companies that wish to obtain advice without having a process in place yet, or with a process that is to be abandoned altogether. Although the latter will only very rarely happen, it should be taken into account.

Based on this choice, a questionnaire is generated and performed to gather information regarding the situational context. In the area of SPM, the situational analysis has been performed by conducting a questionnaire with a list of all the relevant situational factors as described by Bekkers et al. [1]. To enhance reliability of the data, the questionnaire could be replaced by performing an interview. Similar solutions can be developed for other areas.

Data from interviews that have been held suggest that there is a variety of wishes regarding the amount of effort that companies are willing to put in, in order to obtain process improvement advice. We can distinguish two manners in which companies are willing to provide information regarding their current process:

- Full process information. In the optimal case, companies are willing to provide complete information regarding their current process, deliverables, and situational factors that describe their environment. This means that their entire process needs to be captured in a way suitable for further elaboration. Also, the situational factors need to be captured in some way, either through a questionnaire or by means of an interview. With all data available, the process improvement advice that can be obtained is the most effective. However, capturing the entire process requires significant work from an expert who is able to employ an appropriate modeling technique.
- Only situational factors and maturity information. In many cases, capturing full process information requires too much effort. Therefore, it should be possible to provide a process improvement advice based solely on the situational factors and maturity information. This option implies that the advice does not contain any information on how to implement the advice, but only what should be implemented.

If a company is willing to provide full information regarding (part of) their process, the process should be modeled by an expert, either internal or external. The resulting model should contain detailed information regarding both the process as well as the deliverables. Therefore, process-deliverable diagrams (PDDs) are a very suitable technique for this purpose. Vlaanderen et al. [22] show how PDD's can be used to model an SPM process and to capture the current maturity level of a company's product management process.

2.2 Analysis of Need

The next phase takes the situational factors and the list of implemented capabilities from the first phase as input, after which it determines how the current process could be improved. In the domain of Software Product Management, this phase has already been described by Bekkers et al. [1] in the form of the situational assessment method, but it will be summarized here for the sake of completeness (see Figure 3). The need analysis consists of three activities; (1) construction of the current capability profile, (2) calculation of the optimal capability profile, and (3) calculation of an 'areas of improvement' matrix. The first of these three consists of translating the results from the initial maturity assessment into a form usable for further calculation.

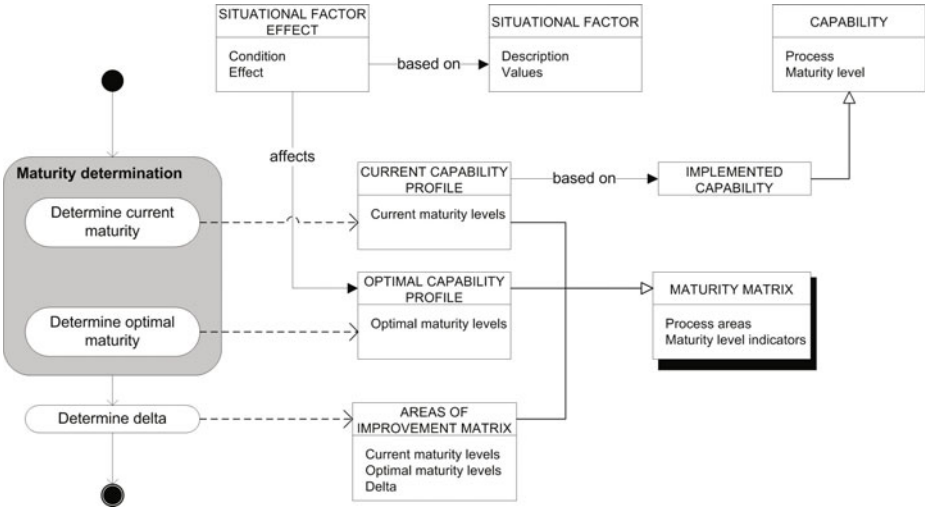


Fig. 3. Analysis of Need

Dark grey: actual maturity level

Focus Area	Maturity Levels										
Title	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Requirements Management											
Requirements Gathering		A		B	C		D	E	F		
Requirements Identification			A			B		C			D
Requirements Organizing				A		B		C			

Light grey: optimal maturity level

Fig. 4. Example Areas of Improvement Matrix

The second activity is somewhat more complex. The optimal capability profile is determined by a set of situational factor effects. Several situational indicators have an associated effect. By applying all applicable situational factors effects, an optimal capability profile is obtained that is customized for the current company.

The current capability profile and the optimal capability profile are then combined into an Areas of Improvement matrix. This is again a capability matrix, with both previous matrices integrated into it. Between the two matrices, a gap can exist, which can be called the delta. This delta indicates the capabilities that need to be implemented, in order to arrive at the optimal maturity level. An example of such an Areas of Improvement matrix within the Software Product Management domain is shown in Figure 4. The actual delta is the light-grey areas, as this depicts the difference between the actual and the optimal maturity level. This set forms the basis for the next phase in the process of method improvement.

What is important in the context of this phase is the fact that users can vary in the rigidity that they demand from the method engineering process. Some wish only a partial improvement for a specific area, while others wish to improve their process to the maximum maturity level suggested for them. As stated before, evolutionary improvement is in many cases more prone to success than revolutionary change. This implies that it should be possible to provide improvements in the form of a roadmap when the process is changed rigorously.

2.3 Selection of Process Alternatives

For the next phase, each missing capability has to be connected to a method fragment that implements the capability. The capabilities that a method fragment implements can be used as an attribute during the initial selection of method fragment candidates. As will be described later on, both process fragments as well as deliverable fragments can implement capabilities. For this reason, capability is an effective first classifier of a method fragment in the method base.

For further classification, we reuse the situational factors described by Bekkers et al. [2]. Since many fragments will be applicable in any situation, it does not make sense to describe each fragment by all factors. This would also pose a problem when the list of situational factors would change. Therefore, situational factors should only be used to indicate restrictions on the use of the fragment. The combined set of indicators for a specific fragment forms its second classifier, situation.

A third classifier of method fragments is their rating. Through the feedback of users, method fragments are rated on several aspects, such as effectiveness, complexity, etc. Method fragments with a very low rating can be ignored in most cases, while in other cases method fragments with a high rating are selected over similar method fragments with a low rating. A simple example of a method fragment with its describing attributes can be found in table 1. It is based on a prioritization technique used within the SPM domain.

Table 1. Example Method Fragment with Attributes

Wiegers' Prioritization Matrix		
Capabilities	**Situation**	**Rating**
– Internal Stakeholder Involvement – Prioritization Methodology – Customer Involvement – Cost Revenue Consideration – Partner Involvement	– # of requirements < 50 – Partner involvement >= medium	– Ease of use: 8/10 – Satisfaction: 6.5/10

Although processes, capabilities and situational factors form a very solid ground for method fragment selection, we need to take into account that we are dealing with processes in which humans are involved. This means that the resulting process needs to fit with the preferences of the people involved in it.

These people need to be able to express these preferences during the selection of alternative method fragments. The results from interviews have indicated that product managers are not always willing to accept suggestions made to them by a machine [21]. Therefore, the process should allow for differences in the amount of freedom that is provided. While it is generally a good idea to suggest one specific method fragment per capability, users should be at liberty to select another. This 'freedom-of-choice' has serious consequences for the OME. In order to make the freedom given to users useful, they need to be provided with a sufficient amount of information for them to base their decision on.

The first source of information for this is the method fragment itself. Since every method fragment can be displayed in the form of a PDD, users can use this diagram to form an initial mental image of its implications. This is possible since all related activities and deliverables are readily available in the method fragment. However, in addition to this, we also identified a need for more sources in the form of experience reports. Experience from people in similar situations is highly valued, and would thus be a valuable addition to the process.

Based on all of the sources of information combined, users should be able to make a valid and well-argued choice regarding the method fragments that should be selected, and thus regarding the changes that should be made to the existing process.

2.4 Creation of Improvement Roadmap

After the improvements have been selected, the process of embedding or implementing the process advice varies depending on the amount of information that a company has provided. The possibilities are limited when only maturity information is known, in contrast with the field of opportunities when full process information is given. In any case, the initial part of the process can be the same for both situations, as this regards the elaboration of the chosen solution into steps. Steps are needed since solutions will in many cases be too large for implementation in one iteration. An evolutionary approach has more chance of success as it will likely yield a higher acceptance due to smaller, incremental changes.

The splitting of solutions into steps is subject to several conditions. Solutions cannot be split into steps randomly. The major reason for this is that we need to take dependencies into consideration. If a company wants to increase the maturity level of its requirements gathering process from A to C (see Figure 4, it does not make sense to implement automation before centralized registration. Instead, the first step should be to implement the activity related to level B, followed by an iteration in which level C is implemented.

In most cases, several capabilities can be implemented at the same time. However, to make iterations or steps more successful, it is probably wise to make sure that each step has some sort of goal, or a theme. This ensures a set of changes that is coherent. This way, the change-process seems less chaotic to the employee. This is important, as he or she will be the one performing the new process.

After the roadmap has been presented to the user and has been accepted, the implementation of it can start. In case that only maturity information is available, this process is fairly straightforward, as little support can be given. The changes that have been proposed need to be implemented in the company manually. In order to guide this, process descriptions and templates related to the advice are provided.

If full process information is available, then this process is considerably more complex. This part encompasses the most complex asset of method engineering, namely the assembly of method fragments. For each step, the selected method fragments need to be integrated with the existing process. As this is a difficult task, it is probably best to do this fragment by fragment.

A problem with this segmented approach, however, is the risk that some parts of the process get changed multiple times. This is unwanted, as this can lead to confusion among the people that need to perform the process. Therefore, already during the creation of the roadmap, the system should make sure that no such situations occur. This is also another argument for the statement that method fragments should be kept as small as possible. By preventing the usage of complex method fragments, the chance of overlap is made smaller, thereby increasing the chance of success of any algorithm that is charged with creating a coherent roadmap.

2.5 Selection and Implementation of Method Increments

After the assembly of the selected method fragments into the original process, the changes can actually be implemented within the company. To facilitate the change, the system can generate and/or update templates based on the original and the new process description (expressed in the PDDs).

If the company's original work documents are available, than they can be updated to reflect the new deliverables within the process. During this step, original data should be maintained while new columns, sections, formulas, etc. are added to the documents.

In case deliverables are not available, templates can be generated based on the generated process description. The generated templates should be directly usable within the new process.

In addition, the system generates full process descriptions with explanations of all steps, deliverables and roles. These descriptions aid the process owner during the implementation of the process in the company.

3 Online Method Engine

The method engineering approach described above does not adhere to the Method-as-a-Service philosophy unless it allows users to perform the created method to some extent online. This means that, instead of using various local software tools, the workflow and the deliverables are embedded in an online platform, which we will refer to as the OME. The method modification aspect

is essential for improving the effectiveness of processes, but the method execution aspect ultimately allows major improvements in the efficiency of these processes by taking away a large share of the burden of maintaining a complex IT infrastructure.

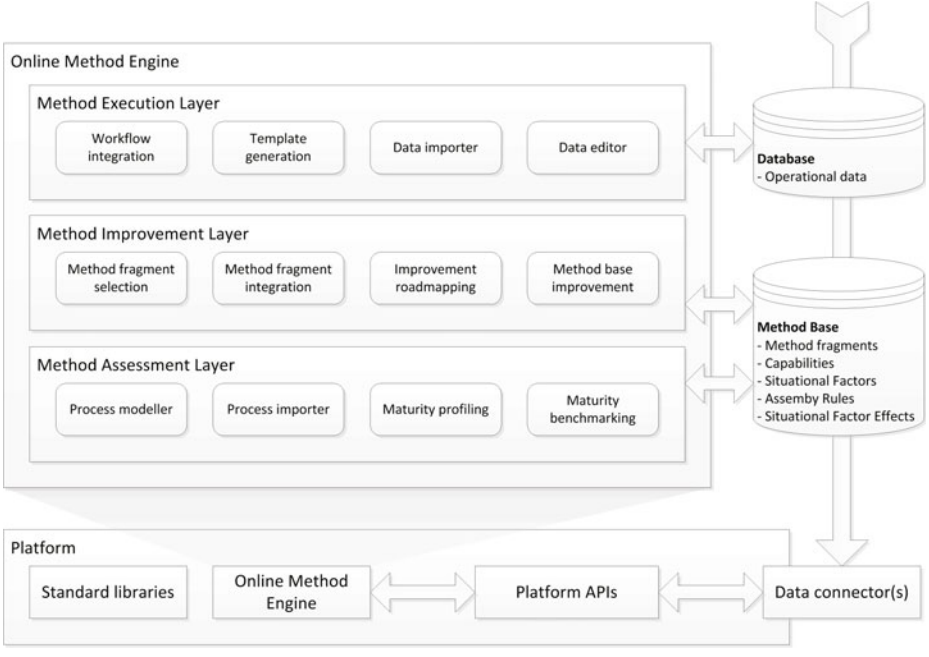


Fig. 5. Online Method Engine

We depicted our vision on the OME in Figure 5. At the bottom, the development platform is depicted. As will be described later, this refers to Google AppEngine in our approach. On top of the platform the OME is shown, with the three functional layers described throughout the previous section. Each layer contains several functional components, shown by the rounded boxes. On the right hand of the figure, two databases are shown; one for the method engineering related data such as method fragments and situational factors, and one for the method execution related data such as requirements and planned releases. These database are connected to the system using separate data-connectors, which allow connection to any suitable database, ranging from a MySQL database to the database of a third-party requirements engineering tool.

Such an approach implies an integration of the functionality to describe a process, assess the process, adapt the process, and then perform the process. In order to be able to do so, method fragments need to be correctly translated into an interface that offers the right set of tools for users to perform their tasks. The creation and usage of templates is a core aspect of this. Such online documents can be placed on any cloud documents solution such as Google Docs. As this

environment is accessible through an API, it can be integrated into any other system, such as the OME.

In addition to the translation of deliverables into documents and the management of these documents, the activities need to be correctly translated. Aspects that need to be taken into consideration here are the correct translation of access rights based on roles, the distinction between automated tasks and user input, the type of interface that is required for a certain set of activities, and the order of the activities, i.e. sequential, simultaneous, or a mix of both.

Currently, these aspects are not all derivable from the PDDs. To solve this, either stricter rules should be applied during the creation of PDDs, or additional models should be created for defining interface, access rights and business process. The former is not a good solution, as this would make the creation of PDD's too complex. The latter is similar compared to model-driven development solutions such as OO-Method [13] and the web-based variant OOWS [12]. This would require the addition of several steps to the process for creating the required models, undermining an important aspect of the OME, namely the fact that it should be simple.

To forgo this problem, an alternative solution could be developed, based on pattern recognition. The idea behind this is that certain patterns will exist in the PDD's of processes and deliverables that can be directly related to correct solutions for the interface. For instance, activities that are performed simultaneously should be connected to a tabbed interface, with a tab for each activity. Activities that are performed linearly can always be displayed as steps, allowing to go back and forward. Such a solution would require no extra effort of the user. However, the possibilities of recognizing patterns are limited and it is very prone to modeling errors. Therefore, a user should always be able to alter the interface for a given process. Alternative interface elements should be provided for this by the system. The same holds for the generation of documents based on deliverables. As it is not always possible to derive the required file-type for a deliverable, the user should have the option to change this manually.

To capture all the requirements of the translation from method description to interface, a meta-model should be defined describing all possible translations for every construct and pattern.

3.1 Information Extraction Using MERL

For capturing processes in the OME, referred to as 'process modeler' in Figure 5, we currently use the tool MetaEdit+. MetaEdit+ is "an environment that allows building modeling tools and generators fitting to application domains" [19]. It lets users define domain-specific meta-models that are used to generate a tool that is suited specifically for creating diagrams based on that meta-model. In this case, the meta-model for PDD has been implemented by defining all constructs (activities, deliverables, etc.) and rules, in addition to the visual aspects of those constructs.

Next to the modeling-capabilities of MetaEdit+, the tool also embeds a transformation language called MERL, or the MetaEdit+ R* Language. This

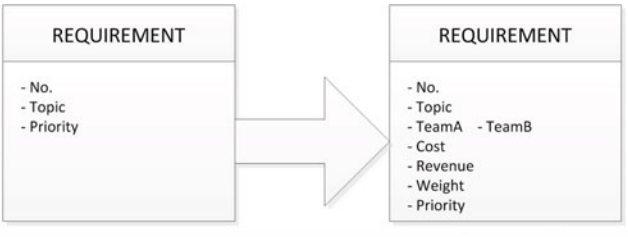
language allows for converting diagrams into any format required. Although the language in itself is not very powerful, some tricks will make any conversion to a textual format such as XML or latex possible.

The language is normally used for code generation, in the context of model-driven development. With such approaches, the solution domain is modeled using a domain-specific language/diagram, after which the diagram is analyzed and converted into source code. In this case, the information stored in the diagrams is used to describe the context / situation of an SPM process, and to assess its maturity (model-driven assessment).

For this research, generators have been written that allow the generation of a filled-in maturity matrix based on all PDDs of a company's process [21]. Combined with the actual diagrams, these pieces of information form a good overview of the maturity of a process, along with its description in terms of activities and deliverables.

3.2 Template Generation

As described earlier, changes made to a process through the OME should be facilitated and supported as much as possible, to ensure the success of the evolution. One technique for doing so is providing automated templates based on



	A	B	C	D
1	No.	Topic	Priority	
2	BR1-100686	Records of emergency equipment inspection & maintenance	low	
3	BR1-100700	Service records of presence sensing systems	high	
4	BR1-100760	Accidental damage to stores, equipment, & property	medium	
5	BR1-101100	Working environment	low	
6	BR1-101101	Outline of warden training program	later	
7	BR1-100542	Safe mechanical & manual handling procedures for movement of materials	critical	
8	BR1-101064	Records of hazard investigations - reportable - non-reportable	medium	

	A	B	C	D	E	F	G	H
1	No.	Topic	TeamA	TeamB	Cost	Revenue	Weight	Priority
2	BR1-100686	Records of emergency equipment inspection & maintenance	10	15	25	130	1	0.29
3	BR1-100700	Service records of presence sensing systems		70	70	130	4	3.30
4	BR1-100760	Accidental damage to stores, equipment, & property	18		18	760	3	3.72
5	BR1-101100	Working environment		100	100	290	3	7.89
6	BR1-101101	Outline of warden training program	100		100	240	1	2.18
7	BR1-100542	Safe mechanical & manual handling procedures for movement of materials	30		30	730	5	9.93
8	BR1-101064	Records of hazard investigations - reportable - non-reportable	15		15	800	1	1.09

Fig. 6. Example of an incremental template

the deliverables of a process. In the case of minor changes, changes made to a template can be incorporated in the original company documents, preserving any data already existing. We developed a proof of concept, in which we show how a Google spreadsheet can be updated dynamically by changing the meta-model of a method fragment.

In Figure 6, the meta-model of a deliverable of a requirements prioritization activity is presented. This deliverable is a requirement. In the old case, this requirement had three attributes: No., Topic, and Priority. This priority was added based on the personal preference of the one who stored the requirements.

In the new situation, another approach is used to prioritize the requirements. The variables 'Cost', 'Weight' and 'Revenue' are added and used to calculate the priority. Furthermore, the attributes 'TeamA' and 'TeamB' are added to divide the costs (in man days) to the teams.

In the two spreadsheets that are illustrated in Figure 6, the change in the meta-model of the requirements can be viewed. Extra columns have been added to the spreadsheet. In this case, some of the cells in the new template are already filled in. Normally, this is a job for the method user.

To make templates useful in practice, the current information as provided by the method fragment does not suffice. Relations within and between concepts are not clearly defined. This means that, for example, the formula that specifies the value of 'Cost' based on the other attributes (in the example above) cannot be modeled. At this point, we do not yet have a satisfactory solution for this.

4 Conclusions and Further Research

In this paper, we presented our vision on the OME, an online environment that can be used to assess an organization's current processes, create an advice based on this assessment, and align the company's tooling infrastructure with the method improvement by automatically configuring templates and documents.

Although the concept presented in this paper is fairly detailed, the OME that we envision is not yet operational. The complexity of such a system was already known, and this paper only strengthens the idea that we are dealing with an advanced concept requiring a lot of research effort. From this point onwards, each of the areas of the OME needs to be addressed in detail, putting together the puzzle piece by piece. Expertise in several areas will be needed, as each part of the OME has its specific challenges, from linguistic analysis for method assembly to data-optimization for the method base.

Up until this stage, the research effort has mainly been focused on analyzing the current situation and the need, with a focus on the SPM domain. Furthermore, a lot of research effort has been spent on the underlying meta-modeling techniques that are used throughout the system. This leaves the remaining areas of process alternative selection, improvement roadmap creation, and increment selection and implementation open for future research.

An important factor that can never be left out during the elaboration is the fact that the purpose of the OME is the improvement of processes. As a

consequence, we are always dealing with people that bring habits, experiences, and opinions. This should not be overlooked. Doing so would result in a system that is too rigid, forcing people into ways of working that they will not accept, thereby foregoing the purpose of the system. However, if it is done right, than the OME has great potential value. We believe that this solution can increase the maturity of the software industry significantly by providing professionals with the right tools to optimize their processes.

Unfortunately, the detailed OME that is presented in this paper has not been fully validated yet. As no concrete system exists yet, doing so would have involved asking potential users to imagine themselves using such a system. This is a tremendous effort, especially due to the complexity of it, and would likely not have resulted in a valid response. However, as development continues, the user should not be forgotten. Instead, at several points in time, his opinion should be asked and corrections should be made according to it. When done correctly, this will result in a functional online method engineering environment.

References

1. Bekkers, W., Spruit, M., van de Weerd, I., van Vliet, R., Mahieu, A.: A Situational Assessment Method for Software Product Management. In: Proceedings of ECIS 2010 (2010) (accepted)
2. Bekkers, W., van de Weerd, I., Brinkkemper, S., Mahieu, A.: The Influence of Situational Factors in Software Product Management: An Empirical Study. In: IWSPM 2008: Proceedings of the 2008 Second International Workshop on Software Product Management, pp. 41–48. IEEE Computer Society, Washington, DC, USA (2008)
3. Berki, E.: Formal Metamodelling and Agile Method Engineering in MetaCASE and CAME Tool Environments. In: Proceedings of the 1st South-East European Workshop on Formal Methods, SEEFM 2003, pp. 170–188 (November 2003)
4. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information and Software Technology 38(4), 275–280 (1996)
5. Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-modelling based assembly techniques for situational method engineering. Information Systems 24(3), 209–228 (1999)
6. Brinkkemper, S., Saeki, M., Harmsen, F.: A Method Engineering Language for the Description of Systems Development Methods. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 473–476. Springer, Heidelberg (2001)
7. Deneckère, R., Iacovelli, A., Kornyshova, E., Souveyet, C.: From Method Fragments to Method Services. In: Proceedings of EMMSAD 2008 (2008)
8. Guzélian, G., Cauvet, C.: SO2M: Towards a Service-Oriented Approach for Method Engineering. In: Proceedings of the International Conference IKE 2007 (2007)
9. Harmsen, F., Brinkkemper, S.: Design and implementation of a method base management system for a situational CASE environment. In: Second Asia-Pacific Software Engineering Conference (APSEC 1995), p. 430. IEEE Computer Society, Los Alamitos (1995)

10. Harmsen, F.: Situational Method Engineering. Ph.D. thesis, Universiteit Twente (1997)
11. Henderson-Sellers, B.: Process metamodelling and process construction: examples using the OPEN Process Framework (OPF). Annals of Software Engineering 14(1), 341–362 (2002)
12. Pastor, O., Fons, J., Pelechano, V.: OOWS: A method to develop web applications from web-oriented conceptual models. In: Proceedings of IWWOST 2003. Luis Olsina, Oscar Pastor, Gustavo Rossi, Daniel Schwabe, Oviedo (2003)
13. Pastor, O., Insfrán, E., Merseguer, J., Romero, J., Pelechano, V.: OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods. In: Olivé, À., Pastor, J.A. (eds.) CAiSE 1997. LNCS, vol. 1250, pp. 145–159. Springer, Heidelberg (1997)
14. Ralyté, J.: Reusing scenario based approaches in requirement engineering methods: CREWS method base. In: International Workshop on Database and Expert Systems Applications (DEXA), pp. 305–309. IEEE, Los Alamitos (1999)
15. Ralyté, J., Jeusfeld, M., Backlund, P., Kuhn, H., Arni-Bloch, N.: A knowledge-based approach to manage information systems interoperability. Information Systems 33(7-8), 754–784 (2008)
16. Rolland, C.: Method engineering: towards methods as services. Softw. Process 14(3), 143–164 (2009)
17. Saeki, M.: Object-oriented meta modelling. Object-Oriented and Entity-Relationship Modeling 1021, 250–259 (1995)
18. Saeki, M.: Came: The first step to automated method engineering. In: Workshop on Process Engineering for Object-Oriented (2003)
19. Tolvanen, J.P., Rossi, M.: MetaEdit+: Defining and Using Domain-Specific Modeling Languages and Code Generators. In: Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA 2003 (2003)
20. van De Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., Bijlsma, L.: On the Creation of a Reference Framework for Software Product Management: Validation and Tool Support. In: International Workshop on Software Product Management (IWSPM), pp. 3–12. IEEE, Los Alamitos (2006)
21. Vlaanderen, K.: Improving Software Product Management Processes: a detailed view of the Product Software Knowledge Infrastructure. Ph.D. thesis, Utrecht University (2010)
22. Vlaanderen, K., van De Weerd, I., Brinkkemper, S.: Model-Driven Assessment in Software Product Management. In: International Workshop on Software Product Management, IWSPM (2010)
23. van de Weerd, I., Souer, J., Versendaal, J., Brinkkemper, S.: Concepts for Incremental Method Evolution: Empirical Exploration and Validation in Requirements Management. In: Advanced Information Systems Engineering, pp. 469–484. Springer, Heidelberg (2007)

A Deductive View on Process-Data Diagrams

Manfred A. Jeusfeld

Tilburg University, Warandelaan 2, 5037AB Tilburg, The Netherlands
manfred.jeusfeld@acm.org
http://conceptbase.cc

Abstract. Process-Data Diagrams (PDDs) are a popular technique to represent method fragments and their recombination to new adapted method specifications. It turns out that PDDs are at odds with a strict separation of MOF/MDA abstraction levels as advocated by MOF/MDA. We abandon the restriction and specify PDDs by a metametamodel that supports both process and product parts of PDDs. The instantiation of the process side of PDDs can then the used as the type level for a simple traceability framework. The deductive formalization of PDDs allows to augment them by a plethora of analysis tools. The recombination of method fragments is propagated downwards to the recombination of the process start and end points. The hierarchical structure of the product side of PDDs can be used to detect unstructured updates from the process side.

Keywords: method fragment, deductive rule, traceability, metamodel.

1 Introduction

Method Engineering advocates the assembly [12] of adapted information system development methods from a pool of method fragments [3], depending on the development context [4,13]. One technique for recording re-usable method fragments are process-data diagrams (PDDs) [14]. They integrate the complex development process with the development products, typically models, documents, and code. The process part is represented using an extension of UML activity diagrams, while the product part is represented as a UML class diagram, utilizing the part-of construct to represent document or model composition. Further information about the method fragment, such as motivation, goals of the method fragment, examples and literature, is textually represented in a Wiki style (http://www.cs.uu.nl/wiki/bin/view/MethodEngineering/).

The first goal of this paper is to investigate how PDDs can be represented in a deductive system that axiomatizes the re-combination of method fragments to larger fragments, and ultimately, to complete methods. The formalization yields

- rules for detecting incorrect re-combinations
- rules to detect unreachable method parts (not discussed here)
- rules to detect unstructured writes to the product side

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 123–137, 2011.

The second goal is to investigate to what extent the PDDs are capable of supporting the traceability of executions of assembled method fragments. We observe that the process part of PDDs itself is a model subject to instantiation and discuss possible extensions to PDDs to allow a limited, but still useful, form of traceability.

The paper is organized as follows. The subsequent section 2 introduces the constructs of PDDs using an example. Section 3 then relates PDDs to the standard abstraction levels of metamodeling. In section 4, PDDs are are formalized using the deductive ConceptBase metamodeling environment [8]. Finally, section 5 relates the structure of the process part structure of the product part of PDDs, and section 6 interprets the execution of a PDD (i.e. process trace) as an instance of the PDD model.

2 Constructs of Process-Data Diagrams

A PDD provides constructs to denote processes similar to UML activity diagrams, constructs to denote the deliverables and data using a variant of UML class diagrams, and a link construct to combine the two sides.

Figure 1 shows an example PDD. Activities like "Domain modeling" can have sub-activities like "Identify relations". Activities can also be associated with

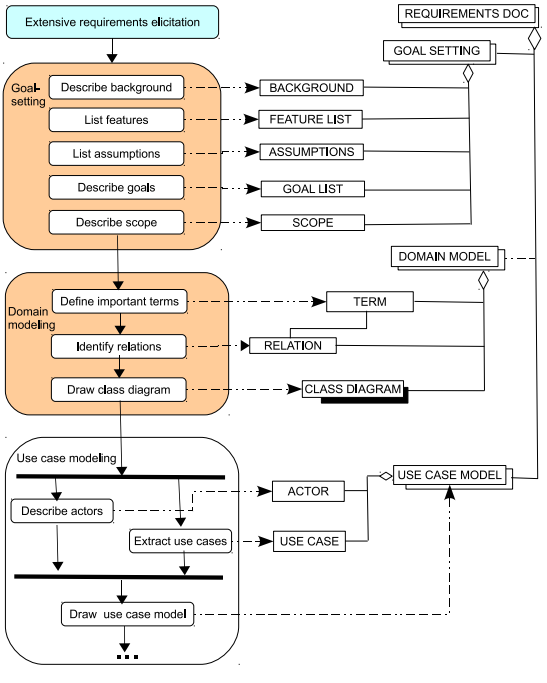


Fig. 1. Example PDD of the Web Engineering Method (excerpted from [15])

agents who perform them (not shown in the figure). Activities exist at various aggregations levels: whole projects, phases, larger activities and individual steps. PDDs consider two types of complex activities. Open activities have explicit sub-activities, and closed activities have sub-activities, but they are not made explicit. Activities are routed via decision nodes ("if then else") and parallel splits. There are also parallel joins, all denoted with UML activity diagrams.

The product part of a PDD is a UML class diagram hierarchically organized via composition associations. Open complex concepts are explicitly decomposed into parts, while closed complex concepts are known to consist of parts but the parts are not shown. The decomposition can be down to individual model elements such as an individual actor in a use case diagram. The hierarchical structure of the product part resembles the hierarchical decomposition of activities into sub-activities. However, there is no strict rule that elements of the process part are matched to elements to the product part that have the same decomposition level, e.g. whole methods matched to the top concept in the hierarchy of data concepts. It is assumed – though not enforced – that the process part of a PDD has a unique start and a unique end. The process and product parts are connected by an output link (dashed arrow in fig. 1).

Method fragments are stored in a method base, for example the Complete Definition Phase method fragment of fig. 1 [14,15]. We shall refer to the method fragment by its name and note that a method fragment is a certain aggregation of an activity, typically covering a phase. The goal of section 4 shall be a logic-based reconstruction of PDDs that allows to formalize syntactic correctness rules for PDDs. The formalization shall also support the automatic recombination of method fragments and a simple form of traceability of a method execution. Specifically, we aim for the following properties:

1. If a method fragment A is defined to be followed by method fragment B then the last activity of method fragment A is followed by the first activity of method fragment B. Note that these two activities can themselves be decomposed. The composition rule then applies to their sub-activities as well.
2. Unstructured writing to data elements should be detectable, i.e. if phase A writes to data elements that are grouped with a complex data element DA, and phase B writes to data elements that are grouped with a complex data element DB, then there should be no activity of A that writes to elements of DB.
3. The origin of actual data elements, i.e. instances of the data element types specified in a PDD should be traceable, i.e. which other data elements were needed in order to produce this data element.

The last function is refers to the execution of a method rather than to its definition in the PDD. We shall introduce the notion of execution as a simple instantiation of a method specified in a PDD.

3 PDDs versus Metamodeling

PDDs combine a product part and a process part. They are in fact workflow models that include the products of the activities inside the workflow model. Still, there is a special clue with PDDs: the product of the activities are usually models, such as a use case diagram. Before formalizing PDDs, we have to understand the abstraction level [6,10] of PDD elements. Subsequently, we use the abbreviations M3 (metametaclass level), M2 (metaclass level), M1 (class level) and M0 (data/execution level) as explained in [2]. Consider the following three statements:

M0/M0 *Bill changes the delivery address of order 453 to "Highstreet 3".*
M0/M1 *Mary defines ORDER as an entity type within ERD-12.*
M0/M2 *Peter proposes EntityType as modeling construct.*

All three statements are about some process executions involving some products. The first statement is a typical element of a business process trace. The products are data elements (abstraction level M0). The trace statement itself is also at M0 level. The second statement is from a modeling activity. The products are model elements (abstraction level M1), but the trace statement itself cannot be further instantiated: it is at the M0 level. Finally, the product part of the third statement is at M2 level, while the statement itself is at M0 level, since the object 'Peter' cannot be further instantiated. The examples show that the abstraction level of the product part characterizes the nature of the process, i.e. whether it is a business process, a modeling process, or a metamodeling process.

The three statements are all excerpts from an execution of a process. The process definition is one abstraction level higher for both the process and the product parts:

M1/M1 *A customer changes the delivery address of an order to a new value.*
M1/M2 *A data modeler defines entity types in entity-relatiship diagrams.*
M1/M3 *A metamodeler proposes constructs of modeling languages.*

PDDs as a notation could represent all three flavors of statements, i.e. the process part of PDDs are always at M1 level and the product part is either M1, M2, or M3. Since method engineers design the workflow models for modelers, a typical PDD is at M1 level for the process part and at M2 level for the product part. Figure 2 puts both the process part (left) and the product part (right) into this MOF perspective. Example PDDs are at M1 level, the data element that they produce are at M2 level.

The OMG-style use of metamodeling strictly separates abstraction levels: they may only be connected via instantiation. The PDD case shows that this strict separation prohibits combined process and product models targeted to method engineering processes. We can still stick to the abstraction levels and the instantiation link between them when regarding only the product part or only the process part.

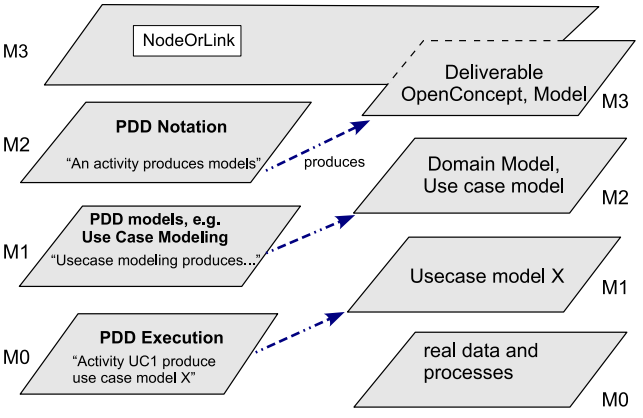


Fig. 2. Putting PDDs into a metamodeling perspective

Another concern for formalizing PDDs is the specification language. As argued, a strict use of MOF leads to a violation of the instantiation rule. The object constraint language (OCL) builds upon the separation of class and instance level. Indeed, one OCL constraint can only link two level pairs [1] and it lacks a fixpoint semantics to follow transitive links in cyclic graphs. We shall therefore use a deductive formalization[1].

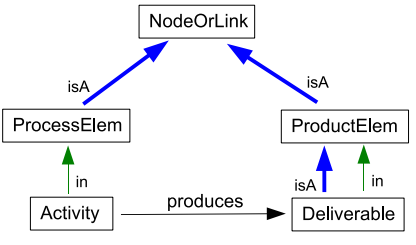


Fig. 3. M3 level for product and process parts

Figure 3 defines the new combined M3 level that can cover both the product and process parts of PDDs. Note that `Deliverable` is both specialization and an instance of `ProductElem`, which itself is a specialization of `NodeOrLink` – the most generic construct of the M3 model used in this paper. Consequently, `Deliverable` can be regarded both as a M3 and M2 object. On the left-hand

[1] Gogolla et al. [5] proposed to represent all abstraction levels into a single instance level and use a generic class level that basically supports the representation of graphs. This representation would consequently allow a use of OCL that is not restricted to just a level pair like M1-M0, M2-M1 etc. As the class level would not contain specific classes, the OCL constraints would be rather complex.

side, `Activity` is an M2 object because it is an instance of the M3 object `ProcessElem`.

4 Deductive Formalization

We use the capabilities of Telos [9] and its implementation in ConceptBase to logically reconstruct the PDD notation and axiomatize its syntax and part of its semantics. ConceptBase implements a dialect of Telos via Datalog-neg, i.e. Horn clauses without function symbols and with stratified negation as failure. This interpretation of a Datalog-neg theory is efficiently computable. We use the following predicates in our formalization:

(x in c) the object x is an instance of the object c, also called the class of x;
(c isA d) the object c is a specialization of object d;
(x m/n y) the object x is associated to object y via a link labeled n; this link has the category m.

Deductive rules are formulated on top of these three predicates, deriving further facts of these predicates. One single base predicate P(o,x,n,y) provides the base solutions for the three predicates [6]. We subsequently formalize PDDs in the frame syntax that aggregates facts of the above three predicates into a textual frame. We use the MOF/MDA abstraction levels in comments to improve readability of the formalizations. They are not part of the formalization. Most of the subsequent formalization is about the structure of PDDs and is represented by facts of the three predicates.

4.1 The Product Part in ConceptBase

The product part of fig. 1 lists models and model elements that are at the M2 MOF level. Hence, to formalize that part, we need to specify its constructs at the M3 level. We formulate it as a specialization of the basic M3 level used in [6].

Constructs of the Product Part of PDDs (M3)

```
NodeOrLink with    {* = (NodeOrLink attribute/connectedTo NodeOrLink) *}
   attribute
    connectedTo: NodeOrLink
end
Node isA NodeOrLink end    {* = (Node isA NodeOrLink) *}
NodeOrLink!connectedTo isA NodeOrLink end
Model isA Node with
   attribute
      contains: NodeOrLink
end
ProcessElem isA NodeOrLink end
ProductElem isA NodeOrLink end
```

```
Deliverable in ProductElem isA ProductElem end
Concept isA Deliverable end
StandardConcept isA Concept end
OpenConcept isA Concept,Model with
  attribute
     contains: Deliverable
end
ClosedConcept isA Concept,Model end
DocumentDeliverable isA OpenConcept end
ModelDeliverable isA OpenConcept end
```

The first constructs are standard constructs for the M3 level: `NodeOrLink` for model elements that are aggregated into models. The second half states that PDD product elements are 'deliverables'. Open concepts are concepts that have other deliverables as parts. The constructs `DocumentDeliverable` and `ModelDeliverable` are introduced to distinguish textual deliverables (e.g. reports) from model deliverables composed of diagrams.

4.2 The Process Part in ConceptBase

The process part in the example of figure 1 is at MOF/MDA M1 level because it can only be instantiated once: its actual execution in the context of some project. Hence, the constructs of the process part to denote such examples are at the M2 level:

Constructs of the Process Part of PDDs (M2)

```
ActivityNode in Node,ProcessElem with
  connectedTo
     next: ActivityNode
end
ActivityDiagram in Model,Class isA Activity with
  contains
     activity: ActivityNode;
     control: ActivityNode!next
end
Phase in Model isA ActivityDiagram end
PDD in Model isA Phase end
PDDLibrary in Model isA PDD end
Agent in connectedTo end
Activity in ProcessElem isA ActivityNode with
  connectedTo
     produces: Deliverable;
     performer: Agent
end
ParallelBranch in ProcessElem isA Activity with
  connectedTo
     branch: ActivityNode
end
ParallelBranch!branch isA ActivityNode!next end
```

```
ParallelJoin in Node isA Activity end
DecisionPoint in ProcessElem isA Activity with
  connectedTo choice: ActivityNode
end
DecisionPoint!choice isA ActivityNode!next end
DecisionJoin in Node isA Activity end
```

Basically, the above definitions are UML activity diagrams augmented with certain extensions for PDDs. Agents are introduced as performers of activities. The control structure of activity diagrams is expressed by the `next` construct of activity nodes (standing for an activity at any aggregation level). We refer to such a link by an expression `ActivityNode!next`. The `produces` construct of `Activity` establishes the link to the data part of PDDs, i.e. the arrows with broken lines in fig. 1.

4.3 Definition of PDD Combination

PDDs follow syntactic rules such as that all activities in the process part must be on the path from the start activity to the end activity (compare also workflow models as presented in [16]). They have a certain semantics such as about the composition of PDDs (method fragments) to larger PDDs or methods. Subsequently we consider our first challenge from section 1: if two PDDs are combined then the combination is inherited downwards to the end and start activities of the participating PDDs. To realize this property, we assume that the basic properties of relations such as transitivity, reflexivity, symmetry etc. are already provided by the ConceptBase system. See [7] for for details. Given these definitions, we specify:

Deductive rules for combining PDDs

```
ActivityDiagram in Model,Class isA Activity with
  reflexive,attribute
     subactivity: ActivityNode
  rule  t1: $ forall ad/ActivityDiagram a/ActivityNode (ad activity a)
                  ==> (ad subactivity a) $;
    t2: $ forall ad1,ad2/ActivityDiagram a/ActivityNode (ad1 activity ad2)
                and (ad2 subactivity a) ==> (ad1 subactivity a) $
end
StartNode in GenericQueryClass isA ActivityNode with
   parameter,computed_attribute
      diagram: ActivityNode
   constraint isStart: $ ((diagram in ActivityDiagram) and
      Adot(ActivityDiagram!activity,diagram,this) and
      not exists a/ActivityNode
            Adot(ActivityDiagram!activity,diagram,a) and
              (a \= this) and :(a next this):) or
               (not (diagram in ComplexActivity)) and (this=diagram) $
end
```

```
Activity in Class with
  rule d1: $ forall a1/ClosedActivity a2/ComplexActivity s/ActivityNode
              (a1 next a2) and (s in StartNode[a2])  ==> (a1 next s) $;
       d2: $ forall a1/ComplexActivity a2/ClosedActivity e/ActivityNode
              (a1 next a2) and (e in EndNode[a1]) ==> (e next a2) $;
       d3: $ forall a1,a2/ComplexActivity e,s/ActivityNode
              (a1 next a2) and (e in EndNode[a1]) and
              (s in StartNode[a2]) ==> (e next s) $
end
```

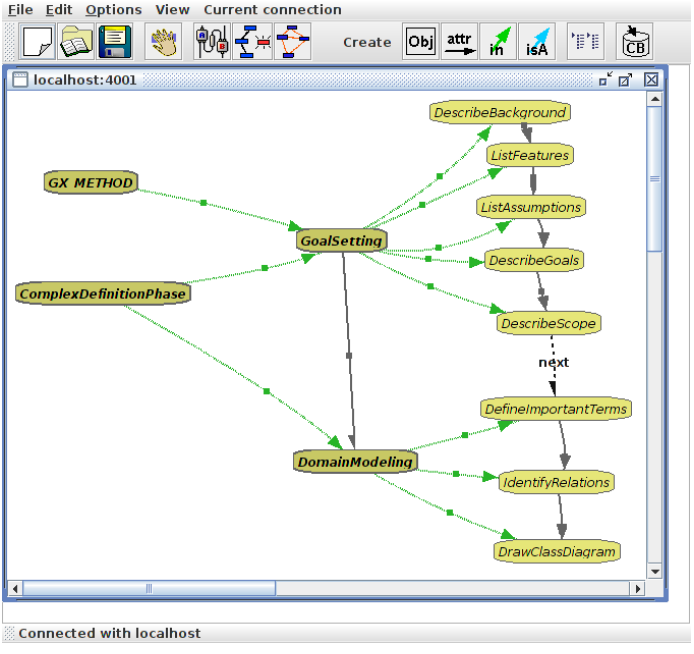


Fig. 4. Combining two PDDs (ConceptBase screenshot)

The concepts `StartNode` and `EndNode`[2] define the first and last activity of a PDD. We also support single activities as (degenerated) PDDs, that are the start and end node of themselves. The main logic is in the deductive rules d1 to d3. The first two special cases are for PDDs that are closed activities. Rule d3 is the general case which takes care that the `next` link is propagated downwards to the start/end nodes. Figure 4 shows a screenshot of an application of the rules. The example is taken from [15] and shows the combination of two PDDs for a Web Engineering Method. The dotted link marked 'next' is inherited via rule d3.

The activity `DescribeScope` is the end activity of `GoalSetting`. The links from left to right are denoting sub-activities. The activity `DefineImportant-Terms` is the first activity of `DomainModeling`. We state `(GoalSetting next`

[2] The concept `EndNode` is defined analogously to `StartNode`.

DomainModeling) denoted by the vertical link between the two. This leads to the deduction of the link (DescribeScope next DefineImportantTerms). If DescribeScope and/or DefineImportantTerms were complex activities themselves, then the 'next' link would be inherited downwards to their start/end activities. Fig. 4 also displays two complex activities GX-Method and Complex-DefinitionPhase. Here GX-Method stands for a library of reusable PDDs and ComplexDefinitionPhase is one phase of the target web engineering method.

5 Detecting Unstructured Data Production

The deductive formalization of PDDs allows to detect certain unstructured accesses from activities to complex data elements. Unstructured writes are characterized by a pattern with two phases that both include activities which write into parts of the same model deliverable aggregating smaller deliverables.

Definition of Unstructured Writing

```
CrossWrittenDeliverable in QueryClass isA ModelDeliverable with
    computed_attribute
       crosswriter: Activity
    constraint
      crossCond: $ exists phase1,phase2/Phase d1,d2/Deliverable
           writer/Activity  (phase1 \= phase2) and
           (phase1 activity writer) and (phase2 activity crosswriter) and
           (writer produces d1) and (crosswriter produces d2) and
           (this contains d1) and (this contains d2) $
end
```

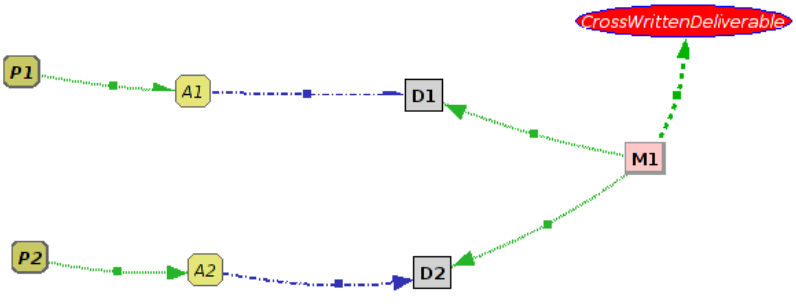


Fig. 5. Cross-written deliverables (screenshot from ConceptBase)

The above query class is returning all model deliverables that are written into by different phases. Hence, in structured PDDs, a phase may not write into a model deliverable that is also written into by another phase. One can argue that this should not always forbidden. Indeed, the formulation as a query class allows a modeler to tolerate violations but still expose them via the query.

Figure 5 shows a generic example of an unstructured writing. The broken links between the activities A1 and A2 and the deliverables D1 and D2 are 'produces' associations. So logically, we have `(A1 produces D1)` and `(A2 produces D2)`. The model deliverable M1 is exposed as instance of `CrossWrittenDeliverable` (oval node in fig. 5).

6 Realizing Traceability

We observed in section 3 that the product part of PDDs is at M2 level, while the process part is at M1 level. We can instantiate both to yield an actual trace of the execution of the process part (M0) linked to data elements at the M1 level. This is a natural relation since modeling is an activity that creates models rather than data from the reality, see also fig. 2.

In the same way the example PDDs are classified into the PDD Process Notation, we can also instantiate them to form a process trace (M0). On the product part, the corresponding instantiation is from a model type (M2) to an example model (M1), e.g. a specific use case diagram. The existing PDD notation only specifies which activity has *produced* a certain product, e.g. a model. It does not specify which products were *required* in order to create it. We extend the PDD notation to include this "input" link as follows:

Extending the PDD (M2)

```
Activity in ProcessElem isA ActivityNode with
  attribute
     retrieves: Deliverable;
     produces: Deliverable;
     performer: Agent
end
```

There is just one additional `retrieves` attribute of `Activity`. The augmented definition now allows us to define coarse-grain traceability on the level of deliverables:

Simple Traceability model (M3-M1)

```
Deliverable in Class with
  rule dr1: $ forall D/Deliverable d/VAR
              (d in D)  ==> (d in DeliverableInstance) $
end
Activity in Class with rule
  ar1: $ forall A/Activity a/VAR (a in A) ==> (d in ActivityInstance) $
end
ActivityInstance in Activity end

DeliverableInstance in Class with
  attribute
    depOnDirectly: DeliverableInstance;
    depOn: DeliverableInstance
```

```
  rule
    depRule1: $ forall d1,d2/DeliverableInstance a/ActivityInstance
     (a [retrieves] d1) and (a [produces] d2) ==> (d2 depOnDirectly d1) $;
    depRule2: $ forall d1,d2,d3/DeliverableInstance
       (d1 depOnDirectly d2) and (d2 depOn d3) ==> (d1 depOn d3) $
end
```

The construct `Deliverable` is at the M3 level. However, we are interested in traceability at the level of example deliverables (M1) such as an example use case model X. To do so, rules dr1 and ar1 ensure that any M1 deliverable is also an instance of `DeliverableInstance`, and that any M0 activity instance is an instance of M1 `ActivityInstance`. This axiomatization allows us to realize traceability *regardless* of the specific PDDs in our library. The rules work with all PDDs.

7 Conclusions

This paper applies a deductive metamodeling approach to the the PDD notation used to represent method fragments. We found that the challenges mentioned in the introduction can be addressed rather easily. The main result is a different one: the rule that only allows instantiation links between abstraction levels is too strict and prevents a proper formalization of PDDs and similar techniques. The rule is neither necessary nor useful. There are some open problems and shortcomings:

- The combination of method fragments fails if there is not a unique start and end activity of the participating method fragments. One may want to support multiple such activities and combine them with decision points (if-then-else) or parallel branches/joins.
- The traceability model neglects the decomposition of deliverables. If a part of a deliverable depends on some part of some other deliverable, then the aggregated deliverables should also depend on each other.
- The formalization is represented by a deductive database, more precisely Datalog with negation. The fixpoint semantics compute the unique minimal Herbrand interpretation under closed-world assumption. This allows direct implementation and use of the formalization but is weaker than a full first-order logic specification.

The formalization is embedded into an existing M3 model. Analysis techniques developed for that M3 model are directly applicable, for example the analysis of connectivity between model elements. The integration with Graphviz allows us to generate diagrams with a reasonable layout (see appendix) from the PDD represesentation in ConceptBase. The re-combination of PDDs is governed by deductive rules that automatically connected the correct ends of the participating PDDs, even if they are deeply decomposed.

Acknowledgments. This paper has been motivated by a challenge formulated by Inge van Weerd when she gave a guest lecture on PDDs in the method engineering course in Tilburg.

References

1. Baar, T.: The definition of transitive closure with OCL – limitations and applications. In: Broy, M., Zamulin, A.V. (eds.) PSI 2003. LNCS, vol. 2890, pp. 358–365. Springer, Heidelberg (2004)
2. Bézivin, J., Gerbé, O.: Towards a precise definition of the OMG/MDA framework. In: ASE 2001, pp. 273–280. IEEE Computer Society, Los Alamitos (2001)
3. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information & Software Technology 38(4), 275–280 (1996)
4. Brinkkemper, S., Saeki, M., Harmsen, F.: Assembly techniques for method engineering. In: Pernici, B., Thanos, C. (eds.) CAiSE 1998. LNCS, vol. 1413, pp. 381–400. Springer, Heidelberg (1998)
5. Gogolla, M., Favre, J.M., Büttner, F.: On squeezing M0, M1, M2, and M3 into a single object diagram. In: Proceedings Tool-Support for OCL and Related Formalisms - Needs and Trends (2005)
6. ISO: ISO/IEC 10027: Information technology - information resource dictionary system (irds) - framework (1990), http://www.iso.org/iso/catalogue_detail.htm?csnumber=17985
7. Jeusfeld, M.A.: Partial evaluation in meta modeling. In: Ralyté et al. [11], pp. 115–129
8. Jeusfeld, M.A.: Metamodeling and method engineering with ConceptBase. In: Metamodeling for Method Engineering, pp. 89–168. The MIT Press, Cambridge (2009)
9. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: Representing knowledge about information systems. ACM Trans. Inf. Syst. 8(4), 325–362 (1990)
10. Object Management Group: Meta object facility (mof) core specification (2006), http://www.omg.org/spec/MOF/2.0/PDF/
11. Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.): Situational Method Engineering: Fundamentals and Experiences, Proceedings of the IFIP WG 8.1 Working Conference, Geneva, Switzerland, September 12-14. IFIP, vol. 244. Springer, Heidelberg (2007)
12. Ralyté, J., Rolland, C.: An assembly process model for method engineering. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 267–283. Springer, Heidelberg (2001)
13. Rolland, C.: Method engineering: Trends and challenges. In: Ralyté et al. [11], p. 6
14. van den Weerd, I.: Advancing in Software Product Management - A Method Engineering Approach. Ph.D. thesis, Utrecht University (2009)
15. van den Weerd, I.: Guest lecture on meta-modeling for method engineering (2010)
16. van der Aalst, W.M.P., van Hee, K.M.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2002)

Appendix: Graphviz Visualization

The PDD visual notation can be approximated by converting the PDD representation of ConceptBase into a format that can be processed by Graphviz[3]. Figures 6 and 7 show two example PDDs excerpted from ConceptBase and layed out by Graphviz. Figure 8 aggregates them with others to a whole phase.

The complete specification of the formalization including the Graphviz integration is available on `http://merkur.informatik.rwth-aachen.de/pub/bscw.cgi/3045636`. It contains also a couple of additional analysis queries that were not included in this paper due to space limitations.

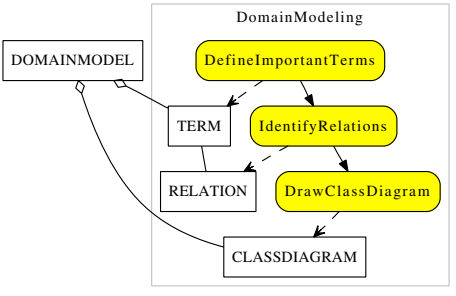


Fig. 6. Domain Modeling PDD layed out by Graphviz

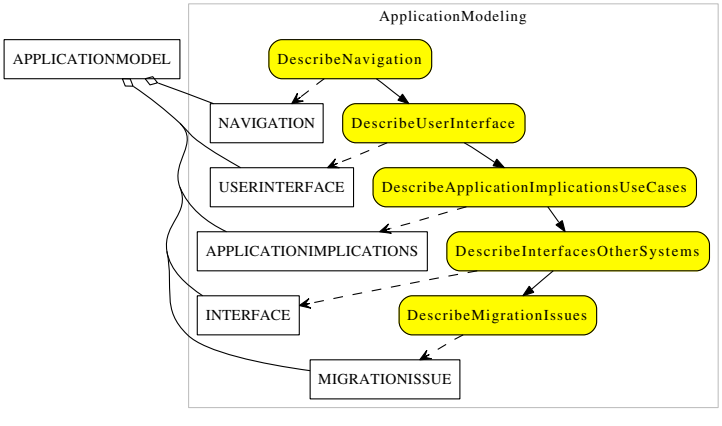


Fig. 7. Application Modeling PDD layed out by Graphviz

[3] See `http://graphviz.org`

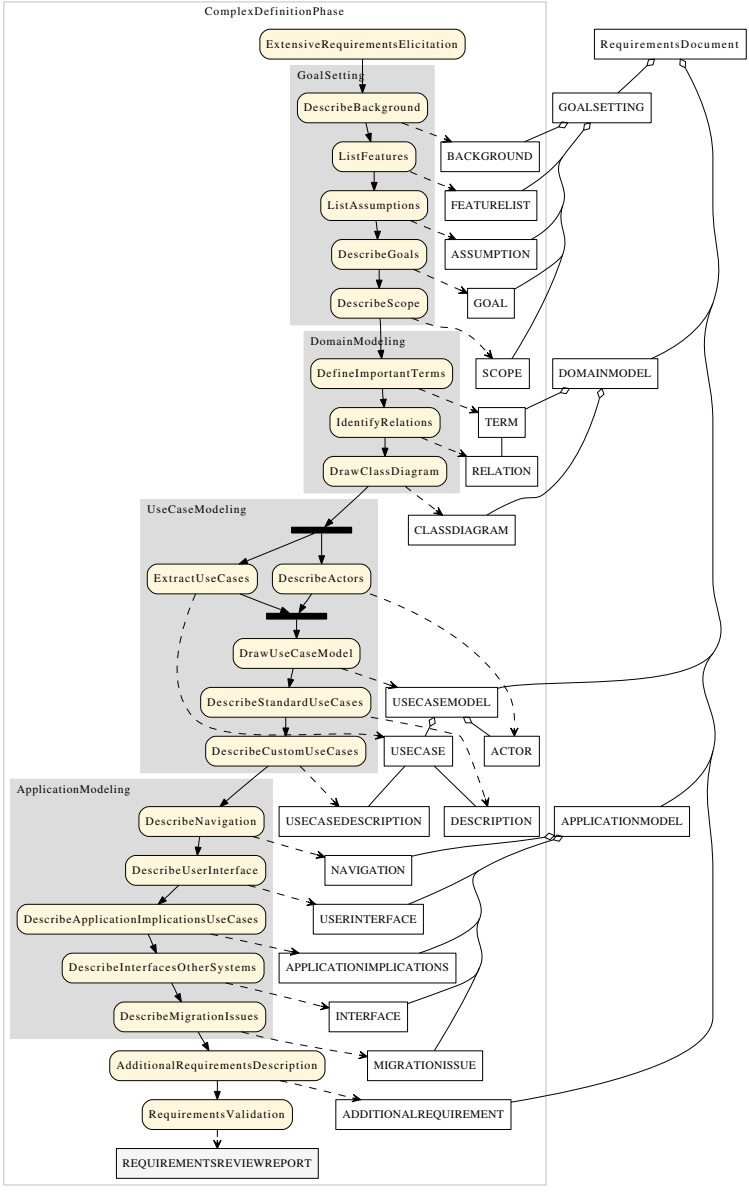


Fig. 8. Graphviz visualization CompleteDefinitionPhase

Turning Method Engineering Support into Reality

Mario Cervera, Manoli Albert, Victoria Torres, and Vicente Pelechano

Centro de Investigación en Métodos de Producción de Software,
Universidad Politécnica de Valencia, 46022 Valencia, Spain
{mcervera,malbert,vtorres,pele}@pros.upv.es

Abstract. The Situational Method Engineering (SME) discipline emerged two decades ago to face up to the challenge of the in-house definition of software production methods and the construction of the corresponding supporting tools. However, nowadays most of the existent proposals only focus on one of the phases of the SME lifecycle. In order to fill this gap, in this paper we present a methodological framework that equally encompasses two of these phases, which refer to the method design and implementation. In order to support them in an effective manner, we advocate for the use of the Model Driven Development (MDD) paradigm. Applying these ideas, the framework has been defined on top of a MDD infrastructure based on meta-modeling and model transformation techniques. In addition, we provide implementation details of the framework in an Eclipse-based modeling platform, namely MOSKitt.

Keywords: Method Engineering, Model Driven Development, CAME Environment, Eclipse, MOSKitt.

1 Introduction

Software Production Methods (hereafter simply methods) are organized and systematic approaches for software development, which can adequately govern the disciplined execution of real software development projects, and are composed, inter alia, of structured and integrated sets of activities, work products and roles. Since the definition of a universally applicable method has for long been considered unattainable, it is necessary to find solutions that enable the in-house specification of methods adapted to specific context needs and the construction of the corresponding supporting tools. Up to now, the SME discipline seems to be the most promising alternative to supply this need.

The SME discipline constitutes a sub-area of a broader field called Method Engineering (ME). Specifically, within the ME (and SME) field, method and software engineers mainly deal with (1) the definition of methods (method design) and (2) the construction of the supporting software tools (method implementation)[1]. Therefore, proposals aimed at supporting ME should cover these two phases of the ME process. However, most of the ME proposals existing in the literature (and their corresponding

[1] Other tasks such as the analysis of the method requirements and the validation of the method are also part of the Method Engineering discipline but are outside of the scope of this paper. These tasks will be considered in future work.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 138–152, 2011.

tools) only focus on one of them. As examples of this reality we find Computer Aided Method Engineering (CAME) and metaCASE environments. On the one hand, CAME environments generally focus on the method design phase, supporting the specification of project-specific methods for software development. In some cases, these specifications are used for building CASE tools, but with very limited capabilities. On the other hand, the so-called metaCASE environments generally focus on the method implementation, supporting the customization of CASE tools by means of high level specifications. These specifications normally define the modeling languages that are to be supported by the CASE tool and, sometimes, also the process that establishes the order in which these languages must be used. Thus, these specifications are oriented towards CASE tool definition and therefore they do not represent complete software production methods.

In order to provide a more complete proposal, in this paper we propose a methodological framework that equally encompasses the method design and method implementation phases. Combining these two phases brings an important benefit. It increments the method specifications' value in terms of how much functionality is derived from them. That is to say, these specifications are not only used for governing the execution of the software development projects, but also for the construction of CASE tools that support the methods and assist the software engineers in the development of the final systems. To achieve this goal in an effective manner, we find crucial to define an infrastructure that (1) allows the method engineer to define methods that can be applied in real software projects and also (2) (semi)automates the construction of tools that provide adequate support to the specified methods. To successfully face the definition of this infrastructure, we advocate for the use of the MDD paradigm. Thereby, we have defined a MDD infrastructure based on meta-modeling and model transformation techniques that lays the foundations of the methodological framework. Specifically, the meta-modeling techniques are based on the Software & Systems Process Engineering Meta-model (SPEM) [30] and are the means that allow the method engineer to carry out the method design. On the other hand, model transformations (semi)automate the performance of the method implementation. By applying these ideas, we have defined a methodological approach that not only tackles the definition of methods following a widely accepted standard (SPEM), but also proposes to use these definitions for the (semi)automatic generation of tools that provide rich support to the methods (textual and graphical editors, code generators, model transformations, process enactment support, etc.).

The work reported here is an extension of our previous works [7] and [8]. On the one hand, the theoretical part of the methodological framework is analyzed in depth, with a contextualization of the different parts of the framework. On the other hand, the software infrastructure of the framework has evolved by enhancing the way in which engineering tools assist method engineers during the method construction.

Furthermore, as a proof of concept, we also provide details of the implemented framework, which has been developed on top of MOSKitt [21], an Eclipse-based modeling platform whose plugin-based architecture and integrated modeling tools turn it into a suitable platform to support the proposal.

The remainder of the paper is structured as follows. First, section 2 summarizes the state of the art. Then, section 3 provides an overview of the proposal. Section 4 and 5 thoroughly detail the MDD infrastructure and the methodological framework respectively. Finally, section 6 draws some conclusions and outlines future work.

2 State of the Art

The term Method Engineering was first introduced in the mid-eighties by Bergstra et al in [4]. Since then, many works developed both at academia and industry have contributed to this field. In order to underpin its theory, a survey of the last strands in ME is gathered in [17]. In this work, the definition proposed by Brinkkemper et al. in [5] is used to define ME as *the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems (IS).*

Considering this definition, we have found that there are proposals in the ME literature that mainly focus on (1) the design, construction and adaptation of methods (i.e. the method design) while others concentrate on (2) the techniques and tools for supporting such methods (i.e. the method implementation). On the one hand, among the proposals mostly dedicated to method design, we find proposals such as Brinkkemper's [5, 6], Ralyté's [20, 24] or Henderson-Sellers' [15], which tackle the method construction by means of the assembly of method fragments or chunks stored in a method base repository. Examples of tools that fall in this first category are MERET [18], Method Editor [29] and Decamerone [14]. Some of these proposals do support the generation of CASE environments but with limited capabilities. For instance, Method Editor enables the generation of tools that include a series of diagram editors that allow the software engineer to create/manipulate the products specified in the method. However, Method Editor does not support the specification of automated tasks that require the inclusion of a model transformation in the generated tool. Thus, these CASE tools lack code generation capabilities.

On the other hand, there are proposals that mostly focus on the method implementation [10, 12, 28]. These are the so-called metaCASE environments that generally support the construction of CASE tools. Examples of tools that fall in this category are MetaMOOSE [10], KOGGE [28] and MetaEdit+ [19]. For instance, MetaEdit+ [19] provides a specification language (called GOPPRR) that is oriented towards the definition of the abstract syntax of the modeling languages (in [19] called "methods") that need to be supported by the resulting CASE tool. In contrast, in our proposal we provide a full methodology that assist in the definition of complete software production methods by means of the SPEM standard, and also proposes the use of a meta-metamodel (such as GOPPRR) for the definition of the modeling languages that enable the creation of the method products (see sections 3 and 5). In particular, the meta-metamodel that is used in the CAME environment that supports our proposal is Ecore.

After studying all the aforementioned proposals, we have found an important lack of software tools that provide complete support to ME. In this paper, we advocate for the use of the MDD paradigm as a way to improve this situation. In particular, we define a methodological framework that is being implemented in the context of the MOSKitt platform [21] and, by applying MDD techniques, equally supports the method design and the method implementation phases.

3 Overview of the Proposal

In order to provide an overview of the proposal, in this section the methodological framework is briefly introduced. The three phases that compose the framework are: method design, method configuration and method implementation (see Fig. 1).

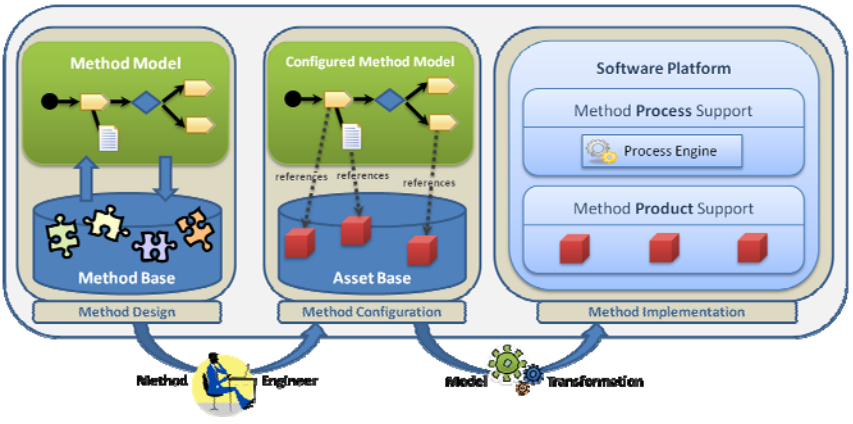


Fig. 1. Methodological framework overview

- *Method design*: during this phase, the method engineer builds the method specification as a model (hereafter the method model) using the SPEM standard [30]. This model can be built from scratch or reusing method fragments stored in a *Method Base* repository that has been implemented following the RAS standard [26]. The built model constitutes a first version of the method that does not include details about the technologies and notations that will be used during the method execution. For instance, the method engineer can specify a generic product called "Business Process Model", without stating in which notation this product will be created when the method is executed.
- *Method configuration*: in this phase, the method model is instantiated with the specific technologies and notations that will be used during the method enactment. This instantiation is achieved by associating tasks and products with editors, metamodels, transformations, etc. that are stored in a repository called *Asset Base* (implemented following the RAS standard). For instance, the product "Business Process Model" can be associated with a "BPMN editor". Thus, the method engineer is indicating that this editor must be included in the generated tool, so that it enables the manipulation of this particular product. The main benefit of separating method design and configuration is that we keep generic definitions of methods (which means that we can take this generic definition and perform different method configurations), stressing the importance of reusability.
- *Method implementation*: in this phase, the method model is used as input of a model transformation that generates the tool support. This tool provides support to the product and process parts of the method[2]. The product support consists of the tools that enable the creation/manipulation of the method products (i.e. the resources associated to the method elements in the previous phase). The process support consists of a process engine that enables the method process execution.

[2] The product part represents the artifacts that must be built during the method execution and the process part consists of the procedures that must be followed to build such products.

4 The MDD Infrastructure

In this section we present the MDD infrastructure that lays the foundations of the methodological framework. As mentioned above, this infrastructure is based on meta-modeling and model transformation techniques.

4.1 Meta-modeling

Meta-modeling has always played a key role in the ME field as it allows the definition at a high level of abstraction of the concepts, constraints and rules that are applicable in the construction of methods. In general, proposals focusing on the method design use meta-modeling as their underlying technique to define methods [6, 18, 20]. Moreover, proposals focusing on the method implementation use these techniques to specify the modeling languages supported by the generated tools [12, 19, 28].

In our proposal we use meta-modeling techniques for the creation of the method model, in particular following the SPEM standard. A study about the applicability of SPEM to ME is presented in [22]. In this work, the authors present some of the SPEM advantages and disadvantages for supporting the method design. Among the SPEM advantages we highlight: (1) wide acceptance in the field of process engineering, (2) good ME process coverage, (3) support to both product and process parts of methods and (4) good abstraction and modularization. Regarding its disadvantages, [22] points out the lack of executable semantics, but proposes to overcome this limitation by using a model transformation to transform the process models into executable representations that can be executed by workflow engines.

In order to provide a more in-depth view on how the SPEM meta-model is used in our proposal, below the structure of the method fragments from which SPEM models can be assembled is presented in detail. In general, in the ME proposals that suggest the use of method fragments, these are obtained by instantiating some class of a meta-model. For instance, in the OPEN Process Framework [11] method fragments are generated by instantiation from one of the top levels classes: Producer, Work Product and Work Unit [17]. Specifically, next subsection details the SPEM classes from which method fragments can be created and, furthermore, it presents a taxonomy that classifies the different types of fragments that are used in the proposal.

Method Fragments. We use the term *method fragment* to denote the atomic element from which methods can be assembled. Other terms to name these atomic elements, such as method chunk, have been proposed in the ME literature [16]. A method fragment can be either a *product fragment* (instances of meta-classes that represent products) or a *process fragment* (instances of meta-classes that represent processes). This differentiation allows us (1) to leverage the separation between product and process specification provided by SPEM[3], (2) to relate one process fragment with many

[3] In order to use the same terminology as the used in the ME field, in our proposal we consider analogous the product-process separation of methods and the SPEM separation between method content and method process.

product fragments, and (3) to reuse one product fragment in the definition of many process fragments.

Attending to the different phases identified in our framework (see section 3), we use a third type of fragment, namely *technical fragment*, term that was first proposed in [13]. In our proposal, these fragments contain the tools that are associated to the products and tasks of the method during the method configuration and that make up the infrastructure of the generated CASE tools.

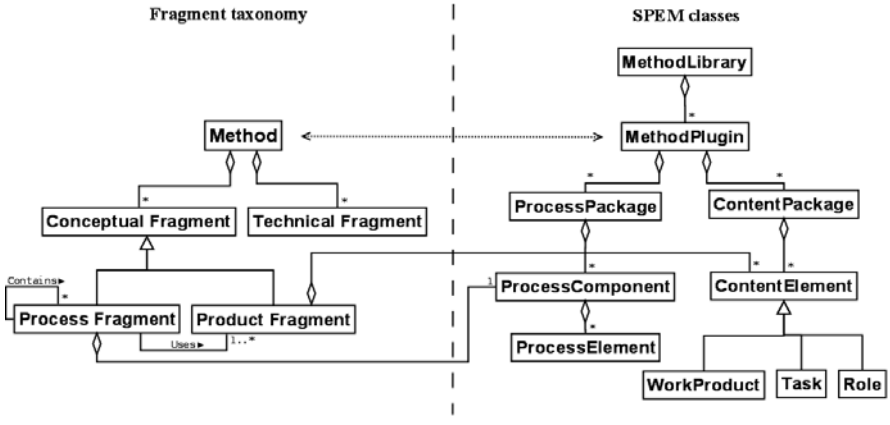


Fig. 2. Relationship between method fragments and SPEM classes

In order to illustrate the hierarchical organization of the various types of fragments, the left side of Fig. 2 graphically presents our fragment taxonomy. In this taxonomy, the new abstract category *conceptual fragment* (also proposed in [13]) is introduced for grouping product and process fragments. Moreover, additional information has been included, e.g. the relationship *Contains* which represents that SPEM processes can contain nested subprocesses, or the relationship labeled as *Uses* which represents that one process fragment can reference from one to many product fragments.

On the other hand, the right side of Fig. 2 shows a simplified view of the SPEM meta-model. In SPEM, a method is represented by a *MethodPlugin*. Each *MethodPlugin* contains both *ContentPackages* and *ProcessPackages*. *Tasks*, *Roles* and *WorkProducts* are stored in *ContentPackages*. Similarly, within *ProcessPackages*, processes are stored as instances of the class *ProcessComponent.*

Note that some of these SPEM concepts have been associated with fragments of our taxonomy. These associations illustrate a containment relationship. For instance, process fragments are associated with one *ProcessComponent.* Thus we are representing that, when process fragments are stored in the repository, they contain a SPEM model that includes one instance of the class *ProcessComponent*. Furthermore, product fragments are associated with *ContentElements*, which represents that these fragments can contain any instances of *Task*, *Role*, and *WorkProduct*.

Finally, even though it has been omitted in Fig. 2, method fragments are defined by a series of properties that enable their later retrieval from the repository. The fragment properties are stored in the manifest file of the RAS asset that embodies the fragment. Specifically, we make use of some of the properties defined in [23]. According to these properties, our method fragments are characterized by:

- *Descriptor*: it contains general knowledge about the fragment. For now, we consider the attributes *origin*, *objective* and *type*. Some examples of valid types in our proposal are *task*, *role* and *work product* for product fragments that contain atomic elements, or *meta-model*, *editor*, *model transformation* and *guide* for technical fragments (see section 5.2).
- *Interface*: it describes the context in which the fragment can be reused. For now, we only consider the attribute *situation.*

4.2 Model Transformations

In the previous subsection we showed that the application of meta-modeling in the ME field is not new. However, we find that the ME approaches that make use of these techniques do not really take full advantage of the possibilities that MDD offers. As stated in [3], "*MDD improves developers' short-term productivity by increasing the value of primary software artifacts (i.e. the models) in terms of how much functionality they deliver*". Following this statement and contrary to what current ME approaches do, we want to leverage models going one step further. Defining the method as a model and considering this model as a software artifact allows us to face the implementation of the CASE tool generation process by means of model transformations.

In particular, these transformations have been implemented in the CAME environment that supports our proposal as a single model-to-text (M2T) transformation using the XPand language [31], which is the language used within the context of the MOSKitt project [21] for that purpose. Further details about this M2T transformation are provided in section 5.3.1 and in [8].

5 The Methodological Framework

In this section, we detail the phases in which the methodological framework has been designed. For each of these phases, we provide first a generic description and then we detail the software infrastructure that has been implemented in MOSKitt to support it.

5.1 Method Design

During the method design the method model is built using SPEM. The construction of this model is performed by means of a combination of two approaches proposed in [24]: (1) the paradigm-based and (2) the assembly-based. In order to illustrate how these approaches are applied in our framework, we use the *Map* process meta-model

proposed in [27]. Following this meta-model, processes are represented as labelled directed graphs with intentions as nodes and strategies as edges between intentions.

The Paradigm-Based Approach. In Fig. 3 we show how the method model is built in our proposal following the paradigm-based approach. The hypothesis of this approach is that the new method is obtained either by abstracting from an existing model or by instantiating a meta-model. This starting model is called the *paradigm model*. Specifically, we build the method models by instantiating a meta-model (i.e. the SPEM meta-model).

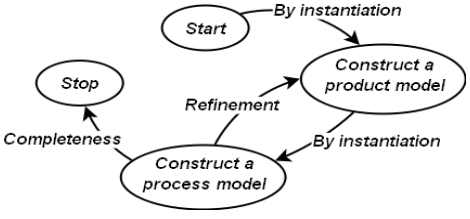


Fig. 3. Paradigm-based approach (adapted from [24])

As shown in the figure, the construction of the method model is performed in two steps: first, the method engineer builds the *product model* (i.e. the products, roles, etc. that compose the SPEM method content). Secondly, the method engineer builds the *process model* (i.e. the process component that composes the SPEM method process). In addition, backtracking to the construction of the product model is possible when building the process model thanks to the refinement strategy.

The Assembly-Based Approach. Fig. 4 shows how the assembly-based approach is carried out in our proposal. This process is followed when the method engineer wants to reuse product or process fragments stored in the Method Base.

As shown in the figure, the method engineer starts by specifying the requirements of the fragments to be retrieved. These requirements are specified as queries that must be formulated by giving values to the method fragment properties (see section 4.1).

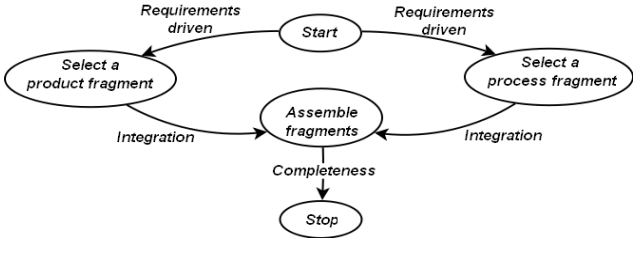


Fig. 4. Assembly-based approach (adapted from [24])

As an example, a query for retrieving a product fragment containing a *task* for *system specification* may include parameters as follows:

Type = 'Task' AND Objective = 'System Specification'

Once the fragments have been obtained[4], the intention "Assemble fragments" must be achieved by means of the "integration" strategy. This strategy consists of the integration of the selected fragments into the method model (considered here as a process fragment of a higher level of granularity). Depending on the type of the fragment this integration varies. For product fragments, the tasks, roles etc. are directly included in a *ContentPackage*. For process fragments, the process elements are included as a subprocess in the method under construction.

Finally, note that during the method design new fragments can be created for their later reuse during the construction of other methods. In order to illustrate how product and process fragments are created, Fig. 5 shows the process that must be followed.

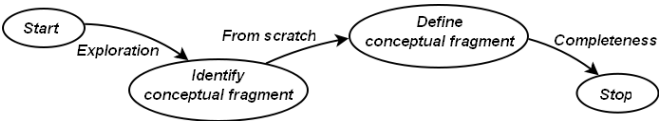


Fig. 5. Conceptual fragment creation (adapted from [25])

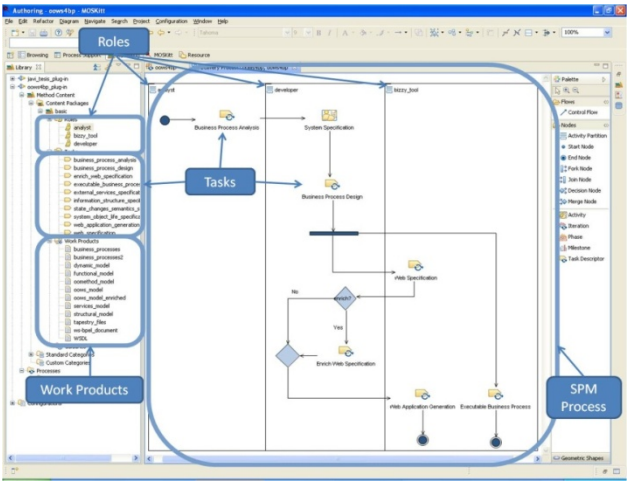


Fig. 6. EPF Composer editor in MOSKitt

[4] Note that if a process fragment is retrieved, then the associated product fragments are automatically selected. This is due to the one-to-many cardinality of the relationship between product and process fragments in figure 2.

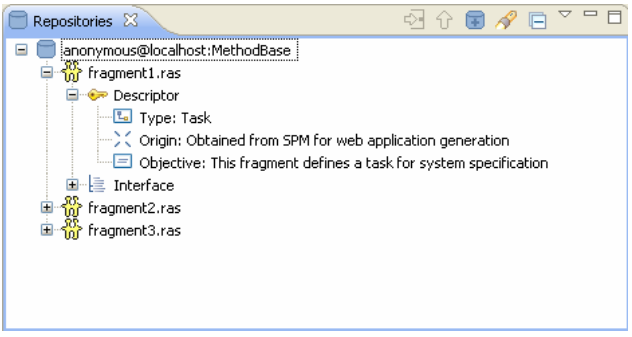


Fig. 7. Repository client connected to the Method Base

First, the method engineer explores the method model to identify the elements that must be included in the conceptual fragment. These elements will be tasks, roles, etc. (for a product fragment) or a process component (for a process fragment). Then, the method engineer defines the fragment by giving values to the fragment properties. Once this process is completed, a RAS asset is created and stored in the Method Base.

Method Design Software Infrastructure. In order to provide software support within MOSKitt to the method design phase, the following tools have been integrated as Eclipse plugins:

- *A method editor*: in order to enhance MOSKitt with the capability of building method models, the EPF Composer (a SPEM 2.0 editor provided in the EPF Project [9]) has been integrated. This editor enables the enactment of the process described in Fig. 3, i.e. it allows method engineers to build SPEM models. In addition, it has been extended so it enables the enactment of the process shown in Fig. 5, i.e. it supports the creation of fragments. In Fig. 6[5], a screenshot of the EPF Composer integrated in MOSKitt is shown.
- *A repository client*: In order to reuse the fragments stored in the Method Base during the construction of the method model, it is necessary to implement a repository client that enables the enactment of the process described in Fig. 4. To do so, the repository client must allow the method engineer to (1) connect to the repository, (2) search and select conceptual fragments and (3) integrate them in the method model under construction. Fig. 7 shows the repository client that has been implemented in MOSKitt as an Eclipse view.
- *A guide to build the method model*: A guide is provided as an Eclipse cheatsheet to assist the method engineer in the performance of the method design phase.

5.2 Method Configuration

In this phase the method model is completed by including details about the technologies and notations that will be used during the method execution. Fig. 8 shows how this phase is performed. In particular, the method engineer specifies the requirements that are used to retrieve a technical fragment from the Asset Base. Once this is done, he/she associates it with a task or product of the method model.

[5] Also available at http://users.dsic.upv.es/~vtorres/moskitt4me/

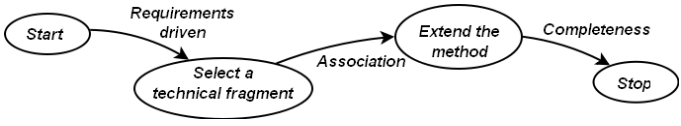


Fig. 8. Process model for asset association

Note that it is possible that no suitable technical fragment is available in the repository. In case the method engineer considers that a new technical fragment must be created, a process similar to the one defined in Fig. 5. is followed. First, the required tool is implemented ad-hoc for the method under construction. For instance, in the CAME environment that supports our proposal these tools are implemented as Eclipse plugins developed using the CAME environment itself. Once the tool is implemented, the method engineer defines the technical fragment by giving values to the fragment properties. Then, a RAS asset is created and stored in the Asset Base.

We detail below the various types of technical fragments that can be stored in the Asset Base, to which elements they can be associated and for which purpose:

- *Meta-model*: meta-models can be associated to method products to specify the notation that will be used in the generated tools for their manipulation (e.g. the "BPMN meta-model" can be linked to the product "Business Process Model").
- *Editor*: textual/graphical editors can be associated to method products to specify the resource that will be used in the generated tools for their manipulation (e.g. a "BPMN editor" can be linked to the product "Business Process Model").
- *Transformation*: model transformations can be associated to tasks of the method. Thus, these tasks will be automatically executed in the final tool by means of the model transformations (e.g. a M2T transformation can be linked to the task "Generate report").
- *Guide*: guides (i.e. text files, process models, etc.) can be optionally associated to manual tasks of the method. These files will be included in the final tool and will assist software engineers in the performance of the tasks. For instance, a map can be associated to the task "Build Business Process Model" to define as a process model the steps that must be followed to perform the task.

Method Configuration Software Infrastructure. In order to provide software support within MOSKitt to the method configuration phase, the following tools have been integrated as Eclipse plugins:

- *A repository client*: In order to associate technical fragments with elements of the method model, it is necessary to implement a repository client that enables the enactment of the process described in Fig. 8. To do so, the repository client must allow the method engineer to (1) connect to the repository, (2) search and select technical fragments and (3) associate them with the elements of the method. The repository client of Fig. 7 can be reused for this purpose. Fig. 9 shows this repository client connected to the Asset Base.
- *A guide to configure the method model*: A guide is provided as an Eclipse cheatsheet to assist in the performance of the method configuration phase.

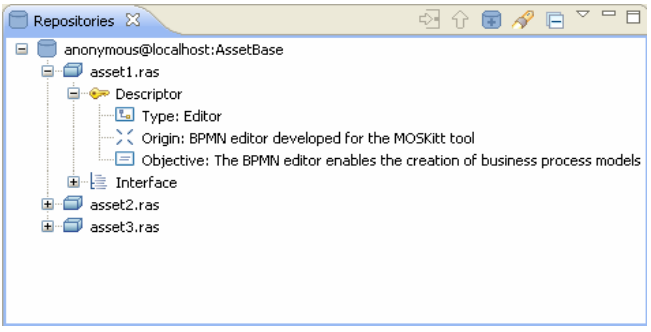


Fig. 9. Repository client connected to the Asset Base

5.3 Method Implementation

During this phase a tool supporting the method is obtained by means of model transformations. This tool is mainly divided into two parts: the dynamic part and the static part (see Fig. 10).

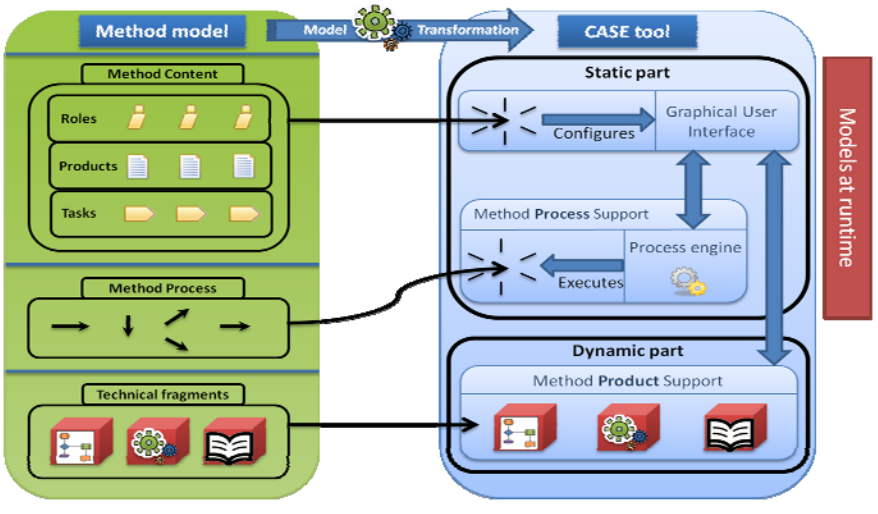


Fig. 10. Transformation mappings

The Dynamic Part. The dynamic part is composed of those elements that are directly obtained from the method model and are, thus, dependent on the specified method. In particular, these elements correspond to the tools that are in charge of providing software support to the product part of the method and make up the infrastructure of the tool (e.g. editors, model transformations, etc.). These tools are specified within the method model as *technical fragments*, which are stored as RAS assets that contain the implementations of the tools (e.g. the Eclipse plugins that implement a graphical editor). Therefore, the model transformation integrates these tools in the generated CASE environment.

The Static Part. The static part is composed of those elements that are always included in the final tool and, thus, their implementation is independent of the method. Even though the implementation of these components does not depend on the method model, they need to use this model at runtime[6]. Specifically, two components make up this part:

- The process engine: this component provides support to the process part of the method. It is always included in the generated tools and is in charge of the execution of the *method process* part of the SPEM model[7]. This execution conducts the orchestration of the different tools that allow the creation/manipulation of the method products (i.e. the technical fragments).
- The graphical user interface (GUI): the GUI is composed of those elements that make up the visual representation of the tool and allow software engineers to execute method instances by means of the process engine. The GUI of the generated CASE tools does not directly depend on the method model (so, they always have the same look & feel) but it uses the *method content* part of the SPEM model to configure itself. For instance, depending on the role selected by the user, the GUI filters its content to show only the products and tasks that the user is in charge of.

Method Implementation Software Infrastructure. In order to provide software support within MOSKitt to the method implementation phase, the following tool has been implemented and integrated as an Eclipse plugin:

- *A M2T transformation:* this transformation obtains the tool that supports the method specified in the method model. This tool corresponds to a MOSKitt reconfiguration that only contains the required Eclipse plugins to support the method (i.e. the plugins contained in the technical fragments[8], the process engine and the Eclipse views that compose the GUI). In order to build this MOSKitt reconfiguration we make use of the Eclipse Product Configuration files (`.product` files). This type of files gathers all the required information to automatically generate an Eclipse-based tool such as MOSKitt. So, considering that this tool is obtained from a `.product` file, the model transformation has been implemented as a M2T transformation. This transformation takes as input the model resulting from the method configuration phase and generates a `.product` file through which the final tool is automatically generated.

6 Conclusions and Future Work

In the ME field it is still unclear how to combine different subareas into a whole in order to define more complete proposals. As examples of this reality we find CAME and metaCASE environments, which either focus on the method design or the method

[6] Runtime in this context refers to the method execution in the generated CASE tool.

[7] SPEM does not have executable semantics. Therefore, a mapping between SPEM and an executable language is needed here. We are planning to tackle this issue in the future.

[8] The dependencies of these plugins must also be included. We are planning to tackle dependencies management in the future.

implementation phases of the ME process. In this work, we have detailed the different steps of a methodological framework that adequately covers these two phases. For this purpose, the proposed framework applies an MDD approach, tackling the method design by means of meta-modeling techniques based on the SPEM standard and the method implementation by means of model transformations.

The presented framework is being defined and implemented within the context of the MOSKitt project. This project constitutes a jointly work developed by the *Conselleria de Infraestructuras y Transporte* and the *Centro de Investigación en Métodos de Producción de Software* to develop a CASE tool to support the gvMétrica method. There is a big community involved in the project, ranging from analysts to end users, which are in charge of validating each new release of the tool. This setting constitutes an adequate environment to validate our proposal. In fact, in the near future we are planning to integrate our prototype into a MOSKitt version in order to use it for the definition of gvMétrica and the construction of the supporting tool.

Regarding future work, we are working on the improvement of the CAME environment that supports our proposal. For instance, we are planning the integration of a process engine such as Activiti [1]. Furthermore, we are concerning with one of the big challenges of ME [2], which deals with the variability of methods at modeling level and runtime. Providing support to variability will allow stakeholders to dynamically adapt methods and their supporting tools to changes that occur during method execution.

References

1. Activiti, `http://www.activiti.org/`
2. Armbrust, O., Katahira, M., Miyamoto, Y., Münch, J., Nakao, H., Ocampo, A.: Scoping Software Process Models - Initial Concepts and Experience from Defining Space Standards. In: ICSP, pp. 160–172 (2008)
3. Atkinson, C., Kühne, T.: Model-Driven Development: A Metamodeling Foundation. IEEE Software 20, 36–41 (2003)
4. Bergstra, J., Jonkers, H., Obbink, J.: A Software Development Model for Method Engineering. In: Roukens, J., Renuart, J. (eds.) Esprit 1984: Status Report of Ongoing Work. Elsevier Science Publishers, Amsterdam (1985)
5. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information and Software Technology 38, 275–280 (1996)
6. Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-Modelling Based Assembly Techniques for Situational Method Engineering. Inf. Syst. 24, 209–228 (1999)
7. Cervera, M., Albert, M., Torres, V., Pelechano, V.: A Methodological Framework and Software Infrastructure for the Construction of Software Production Methods. In: International Conference on Software Processes (2010)
8. Cervera, M., Albert, M., Torres, V., Pelechano, V., Cano, J., Bonet, B.: A Technological Framework to support Model Driven Method Engineering. Taller sobre Desarrollo de Software Dirigido por Modelos, JISBD (2010)
9. Eclipse Process Framework Project (EPF), `http://www.eclipse.org/epf/`
10. Ferguson, R.I., Parrington, N.F., Dunne, P., Hardy, C., Archibald, J.M., Thompson, J.B.: MetaMOOSE - an object-oriented framework for the construction of CASE tools. Information and Software Technology 42, 115–128 (2000)
11. Firesmith, D.G., Henderson-Sellers, B.: The OPEN Process Framework. An Introduction, p. 330. Addison-Wesley, London (2002)

12. Grundy, J.C., Venable, J.R.: Towards an Integrated Environment for Method Engineering. In: Proceedings of the IFIP 8.1/8.2 Working Conference on Method Engineering, pp. 45–62. Hall (1996)
13. Harmsen, A.F.: Situational Method Engineering. Moret Ernst & Young (1997)
14. Harmsen, F., Brinkkemper, S.: Design and Implementation of a Method Base Management System for a Situational CASE Environment. In: Asia-Pacific Software Engineering Conference, p. 430. IEEE Computer Society, Los Alamitos (1995)
15. Henderson-Sellers, B.: Method Engineering for OO Systems Development. Communications of the ACM 46(10), 73–78 (2003)
16. Henderson-Sellers, B., Gonzalez-Perez, C., Ralyté, J.: Comparison of Method Chunks and Method Fragments for Situational Method Engineering. In: Proceedings of the 19th Australian Conference on Software Engineering, pp. 479–488. IEEE Computer Society, Los Alamitos (2008)
17. Henderson-Sellers, B., Ralyté, J.: Situational Method Engineering: State-of-the-Art Review. Journal of Universal Computer Science 16, 424–478 (2010)
18. Heym, M., Osterle, H.: A Semantic Data Model for Methodology Engineering. In: Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering, pp. 142–155. IEEE Computer Society Press, Washington, D.C (1992)
19. Kelly, S., Lyytinene, K., Rossi, M.: MetaEdit+ A Fully Configurable Multi User and MultiTool CASE and CAME Environment. In: Constantopoulos, P., Vassiliou, Y., Mylopoulos, J. (eds.) CAiSE 1996. LNCS, vol. 1080, pp. 1–21. Springer, Heidelberg (1996)
20. Mirbel, I., Ralyté, J.: Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering 11, 58–78 (2006)
21. MOSKitt, `http://www.moskitt.org/`
22. Niknafs, A., Asadi, M.: Towards a Process Modeling Language for Method Engineering Support. In: CSIE 2009: Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering, pp. 674–681. IEEE Computer Society, Los Alamitos (2009)
23. Ralyté, J., Rolland, C.: An Approach for Method Reengineering. In: Kunii, H.S., Jajodia, S., Sølvberg, A. (eds.) ER 2001. LNCS, vol. 2224, pp. 471–484. Springer, Heidelberg (2001)
24. Ralyté, J., Deneckère, R., Rolland, C.: Towards a generic model for situational method engineering. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 95–110. Springer, Heidelberg (2003)
25. Ralyté, J.: Towards Situational Methods for Information Systems Development: Engineering Reusable Method Chunks. In: Proceedings of the International Conference on Information Systems Development, Vilnius Technika, pp. 271–282 (2004)
26. Reusable Asset Specification (RAS) OMG Available Specification version 2.2. OMG Document Number: formal/2005-11-02
27. Rolland, C., Prakash, N., Benjamen, A.: A Multi-Model View of Process Modelling. Requirements Engineering Journal 4(4), 169–187 (1999)
28. Roger, J.E., Suttenbach, R., Ebert, J., Süttenbach, R., Uhe, I., Uhe, I.: Meta-CASE in Practice: a Case for KOGGE, pp. 203–216. Springer, Heidelberg (1997)
29. Saeki, M.: CAME: The first step to automated method engineering. In: OOPSLA 2003: Workshop on Process Engineering for Object-Oriented and Component-Based Development, pp. 7–18 (2003)
30. Software Process Engineering Meta-model (SPEM) OMG Available Specification version 2.0. OMG Document Number: formal/2008-04-01
31. Xpand, `http://www.eclipse.org/modeling/m2t/?project=xpand`

Towards a Method for Engineering Social Web Services

Zakaria Maamar[1], Noura Faci[2], Leandro Krug Wives[3], Hamdi Yahyaoui[4], and Hakim Hacid[5]

[1]Zayed University, Dubai, U.A.E.
zakaria.maamar@zu.ac.ae
[2]Claude Bernard Lyon 1 University, Lyon, France
noura.faci@liris.cnrs.fr
[3]Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
wives@inf.ufrgs.br
[4]Kuwait University, Safat, State of Kuwait
hamdi@sci.kuniv.edu.kw
[5]Alcatel-Lucent Bell Labs, Paris, France
hakim.hacid@alcatel-lucent.com

Abstract. This paper motivates the blend of social computing with service-oriented computing, giving "birth" to social Web services. On the one hand, social computing builds user applications upon the principles of collective action and content sharing. On the other hand, service-oriented computing builds enterprise applications upon the principles of service offer and demand and loose coupling. Thanks to this blend social Web services can operate taking into account with whom they worked in the past and with whom they would like to work in the future. To engineer social Web services, this paper presents a four-step method that addresses several questions related to the engineering exercise. These questions are what relationships exist between Web services, what social networks correspond to these relationships, how to build social networks of Web services, and what social behaviors can Web services exhibit. Experiences dealing with implementing social Web services are, also, reported in the paper.

Keywords: Engineering, Service-oriented computing, Social computing, Web service.

1 Introduction

It is largely known that those responsible for designing and developing enterprise applications appreciate greatly the use of engineering methods while completing their duties. Indeed these methods are road maps that indicate among other things the steps to carry out, the notations to use, the meetings to schedule, and the deliverables to turn in. With the increasing complexity and diversity of today's enterprise applications, different engineering methods (e.g., situational and domain-specific) and scientific fora (e.g., ME'11) have been set up.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 153–167, 2011.

The purpose of these methods and fora is to keep up the pace with the challenges that these applications pose on enterprise applications designers and developers.

Service-Oriented Architecture (SOA) paradigm and its flagship implementation technology namely Web services are among the latest trends in enterprise applications design and development. According to Engels et al., SOA promotes the separation of concerns, information hiding, strong cohesion, and loose coupling [7]. The compliance with SOA principles results in business processes that are flexible and capable of crossing organization boundaries. A Web service is *"a software application identified by a URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts and supports direct interactions with other software applications using XML-based messages via Internet-based applications"* (3WC). When necessary Web services are put together to offer new added-value composite services to users. Different methods and approaches to engineer Web services based-enterprise applications are reported in the literature [10,11,16]. For example, Foster et al. use a model-based approach to verify Web services composition interactions for a coordinated SOA [10]. The adoption of Web services means the ability to support early verification of service implementations against design specifications and that compositions are built with compatible interfaces. Maamar et al. adopt goals to engineer a specific type of Web services that they referred to as capacity-driven Web services [16]. The goals are established to define the roles that capacity-driven Web services play when implementing business applications, frame the requirements that will be put on capacity-driven Web services, and identify the processes in term of business logic that capacity-driven Web services will carry out.

Recently, we started looking into Web services from a social perspective [14]. The purpose is to address some obstacles such as discovery and high-availability that still hinder the widespread acceptance of Web services by IT practioners. By imposing a social perspective on how Web services need to be handled at design- and run-times, we could make Web services (using appropriate tools) establish networks of contacts (i.e., peers) with whom these Web services "feel comfortable" for example to work on common compositions and to recommend for compositions. In this paper, we present our method to engineer $\mathcal{S}$ocial $\mathcal{W}$eb $\mathcal{S}$ervices ($\mathcal{SWS}s$). Briefly this method proceeds as follows: it identifies possible relationships between Web services, builds social networks out of these relationships, and finally, labels Web services as per their roles in these social networks. In this paper the social perspective refers to the social relationships that people come across daily and can be mapped onto relationships linking Web services.

This paper is organized as follows. Section 1 motivates the importance of engineering methods with focus on $\mathcal{SWS}s$ as a case study. Section 2 introduces a running scenario, discusses the overlap between social and service-oriented computing, and provides a brief literature review of $\mathcal{SWS}s$. Section 3 introduces our engineering method. Prior to concluding in Section 5, some technical details on $\mathcal{SWS}s$ are presented.

2 Background

We start with a running scenario that reveals the potential relationships between Web services. Then we introduce the disciplines of social computing and service-oriented computing and how both overlap giving "birth" to $\mathcal{SWS}s$. Finally we review some research works on $\mathcal{SWS}s$.

2.1 Running Scenario

We illustrate the use of social networks of Web services with a scenario related to purchase orders. A customer places an order for a variety of products via *CustomerWS*. Based on this order, *CustomerWS* obtains details on the customer's purchase history from *CRMWS* (Customer Relationship Management). Then, *CustomerWS* forwards these details to *BillingWS* to calculate the customer's bill and subsequently send the bill to *CRMWS*. This latter prepares the detailed purchase order and sends it to *InventoryWS* for order completion. For the in-stock products, *InventoryWS* sends a shipment request to *ShipperWS* to deliver the products to the customer. For the out-of-stock products, *InventoryWS* sends a supply request to the selected *SupplierWS*, which provides *ShipperWS* with the products for subsequent shipments to the customer.

Traditionally the aforementioned Web services (e.g., *CustomerWS*, *ShipperWS*) are discovered and selected without considering or even acknowledging the interactions they had with other peers in previous compositions. Such interactions would have been useful if captured properly using for instance social networks. During the preparation of an order, different relationships are established. The first relationship is the selection that led into identifying *CustomerWS* instead of another Web service for example *OrderSubmissionWS*. Both Web services compete against each other since they do the same job, which is handling customers' online orders. The second relationship is the dependencies between Web services that can be recurrent. *InventoryWS* and *ShipperWS* have constantly participated in several joint compositions. Finally, the third relationship is the high availability of Web services. When *ShipperWS* fails, *DeliveryWS* takes over automatically. If all these relationships had to be captured properly, a Web service would:

- recommend the peers that it likes to **collaborate** with in case of compositions, e.g., *InventoryWS* and *ShipperWS*;
- recommend the peers that can **substitute** for it in case of failure, e.g., *ShipperWS* and *DeliveryWS*;
- and, be aware of the peers that **compete** against it in case of selection, e.g., *CustomerWS* and *OrderSubmissionWS*.

Collaboration, substitution, and competition are relationships commonly found in people's daily life.

2.2 Social Computing Meets Service-Oriented Computing

With the emergence of Web 2.0, social computing is nowadays a hot discussion topic. Some well-known social applications like FaceBook and Twitter exemplify

the successful embracement of Web 2.0. Plus different case studies look into this topic ranging from social computing importance and challenges it raises like privacy to the benefits of adopting social applications by organizations [2].

In [8] social computing is related to applications that support collaborative work (GroupWare) and techniques for modeling, simulating, studying, and analyzing the society (i.e., study the social behavior). Examples of applications include on-line communities and tools, and interactive entertainment and training. Another definition sees social computing as an emerging paradigm that involves a multi-disciplinary approach for analyzing and modeling social behaviors on different media and platforms to produce intelligent applications [12]. Main characteristics of social computing are connectivity, collaboration, and community.

Service-oriented computing is, also, another discipline that attracts the attention of academics and industry people. It aims at bridging the gap between business services and IT services. As stated earlier, SOA and Web services have enabled a new wave of business processes that are loosely-coupled and can cross organization boundaries. The trend of offering business services through the Internet in the form of software services [19] allows the expansion of business services into global markets, ease of access for customers, and increased productivity for companies.

Would there be any overlap between social computing and service-oriented computing? Our answer is affirmative. On the one hand, social computing builds applications upon the principles of collective action and content sharing. On the other hand, service-oriented computing builds applications upon the principle of "I offer services that somebody else may need" and "I require services that somebody else may offer". Service offer and demand illustrate perfectly how people behave in today's society imposing a social dimension on the analysis of Web services. $\mathcal{SWS}s$ can capitalize on their experiences to "identify" with whom they worked in the past and with whom they would like to work in the future.

2.3 Social Web Services in the Literature

$\mathcal{SWS}s$ are at the cross road of service-oriented computing and social computing. Our literature review concluded to a lack of studies that address specific questions related to $\mathcal{SWS}s$ engineering such as how to identify the interactions (or relationships) between Web services and between users and Web services, how to build social networks that capture these interactions, how to navigate through these networks during Web services functioning, and how to maintain social networks in response to changes in Web services. Some other studies adopt $\mathcal{SWS}s$ to illustrate how Web services can help humans interact. For instance, a social service network is proposed in [6]. It integrates Web 2.0 aspects to enrich Web services with semantics. The social aspects, here, are not based on Web services interactions, but how users develop tags out of domain ontologies (i.e., folksonomies) so they assign them to Web services.

Xie et al. suggest a framework for semantic service composition based on social networks [20]. Trust between service providers, service consumers, and services themselves is the social element that is taken into account in this composition.

The framework consists of several modules including semantic extraction and social network construction, social network storage, and trust computing.

Maaradji et al. propose *SoCo* for *So*cial *Co*mposer. *SoCo* advises on the next course of actions to take in response to events such as Web services selection [17]. The advices are built upon the interactions that occur between users and Web services as well as the previously built compositions. *SoCo* consists of different components including social knowledge extraction and modeling, recommendation manager, connection manager, and service repository.

Maamar et al. develop *LinkedWS* as a social networks model for Web services discovery [14]. Different social networks permit to describe the situations in which Web services are engaged for instance collaboration and recommendation. *LinkedWS* stresses out that Web services should not be treated as isolated components that respond to users' queries, only. Contrarily, Web services compete against other, similar Web services during selection, collaborate with other, different Web services during composition, and may replace other, similar Web services during failure despite the competition. Competition and substitution relationships raise an interesting point, which is Web services competing to take part in compositions and at the same time collaborating to support each other during failure. This kind of "behavior" is referred to as coopetition standing for cooperation and competition [3].

Although short, the list of aforementioned research works shows the growing interest in $\mathcal{SWS}s$. To sustain this growth methods for engineering $\mathcal{SWS}s$ are highly deemed appropriate to help highlight the relationships between Web services, the social networks to build with respect to these relationships, and the mechanisms that let Web services use these networks during functioning.

3 Our Engineering Method

Our engineering method consists of four steps that each addresses one of the following questions: what kind of relationships can put Web services in contact, what social networks correspond to these relationships, what components constitute social networks of Web services, and what social behaviors can Web services exhibit. $\mathcal{SWS}s$ that result out of our engineering method are regular Web services that are connected to each other through social networks and exhibit social behaviors based on their role in these networks. We recall that functionality represents the "service" that a Web service offers to users and peers as well.

3.1 Overview

Like any engineering method, our method consists of steps and models that fall into either analysis **or** design phase (Fig. 1). During the analysis phase a service engineer performs two steps. First she establishes relationships between Web services as per the nature of the case study that is under discussion (Section 2.1). Afterwards the service engineer maps the relationships established previously onto social networks, though the mapping is not always one-to-one. During the design phase the service engineer performs two steps as well. First she defines

the characteristics of each social network in terms of number of nodes, types of edges connecting these nodes, and weight formulas for these edges. Finally the service engineer analyzes the social behaviors that Web services exhibit by being part of these networks.

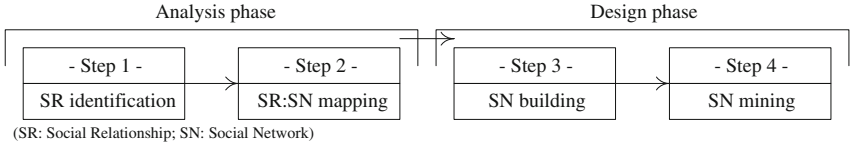


Fig. 1. General representation of $\mathcal{SWSs}$ engineering method

3.2 Step 1: What Relationships Can Put Web Services in Contact?

As stated earlier the objective of this step is to identify the relationships that can exist between Web services. The running scenario has shown three relationships in response to the following cases:

1. Web services that offer semantically similar functionalities like `ShipperWS` and `DeliveryWS`
   - **compete** against each other during selection as only one Web service is considered at a time [4].
   - **substitute** for each other in case of failure so that application operation continuity is maintained [15].
2. Web services that offer separate functionalities like *CustomerWS* and ***InventoryWS*** **collaborate** in the development of new added-value composite services [9].

3.3 Step 2: What Social Networks Correspond to Web Services' Relationships?

As stated earlier the objective of this step is to identify potential social networks that can put Web services in contact. Step 1 resulted in the identification of competition, substitution, and collaboration relationships. Each relationship constitutes a basis upon which a specific social network is developed. As a result, Web services can sign up with three types of social networks: competition, substitution, and collaboration.

- The objective of a competition social network is to make Web services aware of their competitors. This awareness triggers possible enhancements of Web services in case they regularly turn out less competitive at selection time [1].
- The objective of a substitution social network is to make Web services highly available in case failures arise [15]. The potential substitutes are reported in this network, which eases their identification in the future.

– The objective of a collaboration social network is to keep track of all the peers that worked with a Web service on the completion of compositions. The potential collaborators are reported in this network, which eases their identification in the future, as well.

In [14], social networks of Web services are classified into either positive or negative. This is based on the impact that Web services have on each other when they are in the same social network. A positive social network includes Web services that work together since their functionalities are different and hence, can be combined. Contrarily, a negative social network includes Web services that cannot work together since their functionalities are similar and hence, cannot be combined[1]. As per this classification, a competition social network is negative while the other two are positive.

3.4 Step 3: How to Build Social Networks of Web Services?

As stated earlier the objective of this step is to identify the components upon which the social networks of Step 2 are built. We refer to these components as node and edge. In our engineering method nodes and edges correspond to Web services and relationships, respectively.

Competition social network. Fig. 2 illustrates a simple competition social network. Since this network involves similarly functional Web services only, they are all in competition against each other and hence, all connected to each other through bidirectional edges.

To evaluate the weight of a competition edge, which we refer to as *Comp*etition *L*evel (*CompL*, Equation 1) between two Web services ws_i and ws_j, we use the Functionality Similarity Level (FSL) to compare their respective functionalities and the No-Functionality Similarity Level ($NFSL$) to compare their respective non-functional properties (QoS, e.g., reliability level, response time). We assume that the non-functional properties of Web services are defined with the same taxonomy. The use of FSL is shown in Section 4.2.

$$CompL_{ws_i,ws_j} = FSL_{ws_i,ws_j} \times (1 - NFSL_{ws_i,ws_j}) \quad (1)$$

where:

– FSL_{ws_i,ws_j} corresponds to the similarity level between the functionality of ws_i and the functionality of ws_j. This level is determined using existing approaches such as [5] and should be either close to or equal to 1.

[1] Not all Web services can be combined as this depends on factors such as (i) Web services belonging to the same domain for example travel (e.g., *AirBookingWS* and *TaxiBookingWS*) and (ii) data dependencies between Web services (e.g., *OutdoorActivityWS* depending on the feedback of *WeatherForecastWS*).

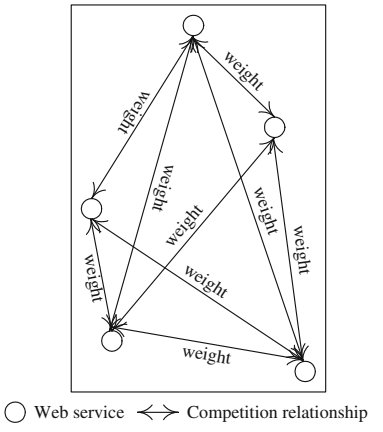


Fig. 2. Illustration of a competition social network

– $NFSL_{ws_i,ws_j} = \omega_1 \times (|P_{ws_{i,1}} - P_{ws_{j,1}}|) + \cdots + \omega_n \times (|P_{ws_{i,n}} - P_{ws_{j,n}}|)$ with $P_{ws_{i,k}}$ is the value of the k^{th} non-functional property of the i^{th} Web service (assumed to be between 0 and 1), ω_k is a weighting factor representing the importance of a non-functional property, and $\sum_{k=1}^{n} \omega_k = 1$.

As per Equation 1 the more the competition level is close to one, the closest ws_i is to ws_j. As a result ws_i threatens the competitiveness capacity of ws_j. We recall that only one Web service can be selected at a time to handle a user's request. A competition social network is useful when a Web service decides to reject processing a user's request for different reasons such as guaranteeing its non-functional properties [13]. This Web service's competition social network permits to identify its competitive Web services so that this request is assigned to one of them upon its approval.

Substitution social network. Fig. 2, also, illustrates a substitution social network after updating the edges' name from competition to substitution. It should be noted that not all edges are bidirectional. Since all Web services in a substitution social network offer the same functionality, any peer can be selected as a potential candidate that replaces a failing Web service.

To evaluate the weight of a competition edge, which we refer to as *Sub*stitution *L*evel ($SubL$) between ws_i and ws_j, we use the Functionality Similarity Level (FSL) and the No-Functionality Similarity Level ($NFSL$) like previously on top of the Reliability Level (RL) that shows how successful ws_i is when it replaces ws_j (Equation 2).

$$SubL_{ws_i,ws_j} = FSL_{ws_i,ws_j} \times RL_{ws_i,ws_j} \times (1 - NFSL_{ws_i,ws_j}) \qquad (2)$$

where:

– FSL_{ws_i,ws_j} and $NFSL_{ws_i,ws_j}$ are defined in Equation 1.

– $RL_{ws_i,ws_j} = \frac{\sum SR_{ws_i,ws_j}}{\sum TR_{ws_i,ws_j}}$, with $\sum SR_{ws_i,ws_j}$ is the total number of Successful Replacements that ws_i made for ws_j (i.e., no failure) and

$\sum TR_{ws_i,ws_j}$ is the Total number of Requests that ws_i received to replace ws_j. If ws_i never replaced ws_j then the substitution level is 0.

Collaboration social network. Fig. 3 illustrates a simple collaboration social network. It is built when at least one composition of Web services is complete. For navigation purposes in a collaboration social network, an entry node is required and represented differently from the rest of nodes (Fig. 3). We refer to this entry node as "focus" Web service. All edges coming out of this "focus" Web service are unidirectional pointing towards other Web services.

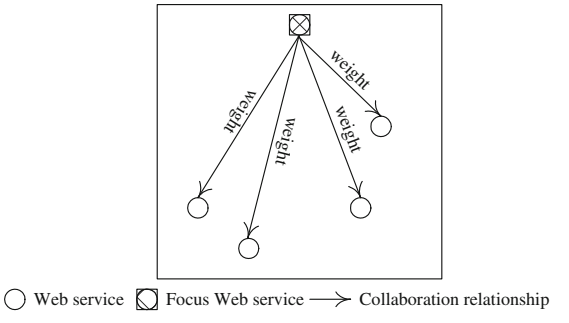


Fig. 3. Illustration of a collaboration social network

To evaluate the weight of a collaboration edge, which we refer to as *Col*laboration *L*evel (*ColL*) between ws_i ("focus") and ws_j, we track the number of times that both Web services participated in joint compositions with emphasis on the total number of compositions that ws_i took part in.

$$ColL_{ws_i,ws_j} = \frac{\sum JC_{ws_i,ws_j}}{\sum TP_{ws_i}} \tag{3}$$

where:

- $\sum JC_{ws_i,ws_j}$ is the total number of participations of ws_i and ws_j in Joint Compositions. $\sum JC_{ws_i,ws_j}$ and $\sum JC_{ws_j,ws_i}$ are equal.
- $\sum TP_{ws_i}$ is the Total number of Participations of ws_i in compositions.

3.5 Step 4: What Social Behaviors Can Web Services Exhibit?

As stated earlier the purpose of this step is to identify the potential social behaviors that Web services can exhibit based on the details that each type of social network (substitution, competition, and collaboration) carries on. Different types of social behaviors exist in real life such as selfish, trustworthy, opportunistic, malicious, vindictive, reliable, etc.

Selfish social behavior. Substitution reveals the selfishness of a Web service when this latter refuses continuously to replace failing peers. However these

peers accept continuously to replace this Web service when it failed. A Web service can use different reasons to back its refusal decisions including "fear" of not meeting its non-functional properties or inappropriateness for replacing a failing peer as per the competition social network ($CompL$ close to 0). To analyze selfishness the substitution relationship between ws_i and ws_j is used as follows, where ws_i substitutes ws_j:

- `If` $SubL_{ws_i,ws_j} = SubL_{ws_j,ws_i}$ `then` the substitution relationship is balanced between both.
- `If` $SubL_{ws_i,ws_j} > SubL_{ws_j,ws_i}$ `then` the substitution relationship is in favor of ws_j, i.e., ws_j did not replace ws_i as many as ws_i did for ws_j; Otherwise the substitution relationship is in favor of ws_i.

Definition 1. ***Selfishness.*** *A Web service ws_i exhibits a selfish behavior if the majority of its substitution relationships with peers are in its favor, i.e., the number of times that $SubL_{ws_i,ws_j} < SubL_{ws_j,ws_i}$ holds, is greater to a threshold T_{SubL}.* □

A Web service that is known as selfish can be ignored by similar peers since these ones cannot count on it when they fail. Corrective actions could be taken by reviewing this Web service's non-functional properties.

Malicious social behavior. Competition reveals the maliciousness of a Web service when it accepts to handle user requests that it receives from other peers, though this Web service is not sure to guarantee its QoS level. Initially these peers declined handling the user requests for reasons listed in the description of the selfish social behavior, and hope that this Web service will not disappoint them. This Web service is reported in these peers' competition social networks.

To analyze maliciousness we introduce a function, called disappointment Dis_{ws_i,ws_j} that tracks of the number of times that ws_i failed in maintaining its QoS level for the user requests it receives from ws_j over the total number of requests that ws_j passed on to ws_i ($Dis_{ws_i,ws_j} = \frac{\sum Fail_{ws_i}(req_{ws_j})}{\sum req_{ws_i,ws_j}}$).

- `If` $Dis_{ws_i,ws_j} = Dis_{ws_j,ws_i}$ `then` the disappointment relationship is balanced between both.
- `If` $Dis_{ws_i,ws_j} > Dis_{ws_j,ws_i}$ `then` the disappointment relationship affects ws_j more than ws_i;

Definition 2. ***Maliciousness.*** *A Web service ws_i exhibits a maliciousness behavior if it is involved in a large number of disappointment relationships with peers, i.e., the number of times that $Dis_{ws_i,ws_j} < Dis_{ws_j,ws_i}$ holds, is greater to a threshold T_{Dis}.* □

Dominant social behavior. Collaboration reveals the dominance of a Web service over a peer when this Web service participates in the compositions of this peer more than what this peer did in the compositions of this Web service.

To analyze dominance the collaboration relationship between ws_i and ws_j is used as follows:

- `If` $ColL_{ws_i,ws_j} = ColL_{ws_j,ws_i}$ `then` the collaboration relationship is balanced between both.
- `If` $ColL_{ws_i,ws_j} < ColL_{ws_j,ws_i}$ `then` the collaboration relationship is in favor of ws_i, i.e., ws_i did participate in the compositions of ws_j more than what ws_j did by participating in compositions of ws_j; Otherwise the collaboration relationship is in favor of ws_j.

Definition 3. ***Dominance.*** *A Web service ws_i exhibits a dominant behavior if the majority of its collaboration relationships with peers are not in its favor, i.e., the number of times that $ColL_{ws_i,ws_j} < ColL_{ws_j,ws_i}$ holds, is greater to a threshold T_{ColL}.* □

4 Implementation

We discuss first, the tools that support service engineers in developing $\mathcal{SWS}s$ and managing their respective social networks and then, report on our experience of dealing with $\mathcal{SWS}s$ in the context of *LinkedWS* project.

4.1 Support Tools

Service engineers who plan to convert Web services into $\mathcal{SWS}s$ use tools to perform this conversion along with other duties such as building $\mathcal{SWS}s$' networks, exhibiting $\mathcal{SWS}s$' behaviors, finding substitutes for failing $\mathcal{SWS}s$, etc.

The first tool called $\mathcal{F}$unctionality $\mathcal{A}$ssessment $\mathcal{T}$ool ($\mathcal{FAT}$) is used by service engineers to establish relationships between $\mathcal{SWS}s$ based on their respective functionalities (Steps 1 and 2). Functionality categorizes $\mathcal{SWS}s$ into either similar or different (Fig. 4). Multiple assessment techniques like those reported in [5] and [18] can be integrated into the $\mathcal{FAT}$. Out of the $\mathcal{FAT}$, two values are obtained: degree of similarity (ds) upon which competition and substitution social networks are built and degree of complementarity (dc) upon which collaboration social network is built.

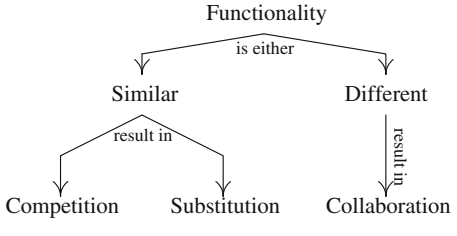


Fig. 4. $\mathcal{SWS}s$ categorization based on functionality

The second tool called $\mathcal{N}$etwork $\mathcal{M}$anagement $\mathcal{T}$ool ($\mathcal{NMT}$) is used by service engineers to build networks of $\mathcal{SWS}s$ and oversee changes in these networks (Step 3). A social network is either built from scratch or extended after adding new nodes/edges to it. For each type of social network discussed in Step 3, the collaboration, substitution, and competition levels ($ColL$, $SubL$, $CompL$) are calculated using the $\mathcal{NMT}$.

The third tool called $\mathcal{N}$etwork M$\mathcal{I}$ning $\mathcal{T}$ool ($\mathcal{NIT}$) is used by service engineers to analyze the details in the three social networks so that the social behaviors of each $\mathcal{SWS}$ are exhibited (Step 4).

4.2 Experience with *LinkedWS*

The development of *LinkedWS* is thoroughly detailed in [14]. We briefly illustrate the use of the $\mathcal{FAT}$ and $\mathcal{NMT}$.

Building upon the work of Min et al. [18], the $\mathcal{FAT}$ associates a Web service (s) with a profile that consists of precondition, input, output, effect, and QoS. The $\mathcal{FAT}$ establishes the degree of similarity $ds(s_i, ds_j)$ (i.e., FSL) (Equation 4) between s_i and s_j using a *matching score* (ms) function defined in Equation 5.

$$ds(s_i, s_j) = \frac{\sum_k w_k \times ms(c_{s_{i_k}}, c_{s_{j_k}})}{\sum_k w_k} \tag{4}$$

where k is the total number of concepts being similar and w_k is the weight associated with the matching score between a pair of concepts. The resulting degree of similarity is between 0 (completely dissimilar) and 1 (maximum similarity).

$$ms(c_{s_i}, c_{s_j}) = f_1 \times f_2 \times f_3 \tag{5}$$

where $f_1 = e^{\alpha l}$ with α as a constant, $f_2 = \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}}$ with β as a smoothing factor, and $f_3 = \frac{e^{\lambda l} - e^{-\lambda l}}{e^{\lambda l} + e^{-\lambda l}}$ with λ as another smoothing factor.

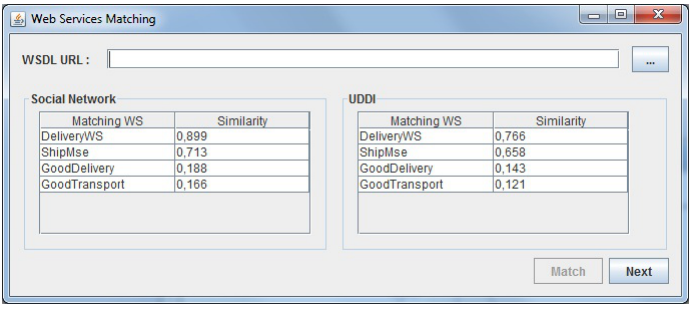


Fig. 5. $\mathcal{WSMC}$ interface for *ShipperWS*

To replace a failing Web service (e.g., *ShipperWS*) using a substitution social network, we implemented the $\mathcal{W}$eb $\mathcal{S}$ervices $\mathcal{M}$atching $\mathcal{C}$omponent ($\mathcal{WSMC}$ that is part of the $\mathcal{NMT}$). The $\mathcal{WSMC}$ is triggered when a Web service fails taking as input the WSDL document of this Web service (Fig. 5). The $\mathcal{WSMC}$ permits to a service engineer to navigate through the substitution social network of a failed Web service.

To build the substitution social network of *ShipperWS* (Fig. 6), we identified manually (using `jUDDI`) some similar peers such as *DeliveryWS*, *ShipMse*, *GoodDelivery*, and *GoodTransport*. Equation 4 assesses the initial weights of the edges that connect *ShipperWS* to these Web services. Once the substitution social network becomes effective these weights are reevaluated as per Equation 2. For instance the total number of Successful Replacements ($\sum SR_{ws_i,ShipperWS}$) that ws_i made for *ShipperWS* is updated where ws_i could for example correspond to *DeliveryWS*.

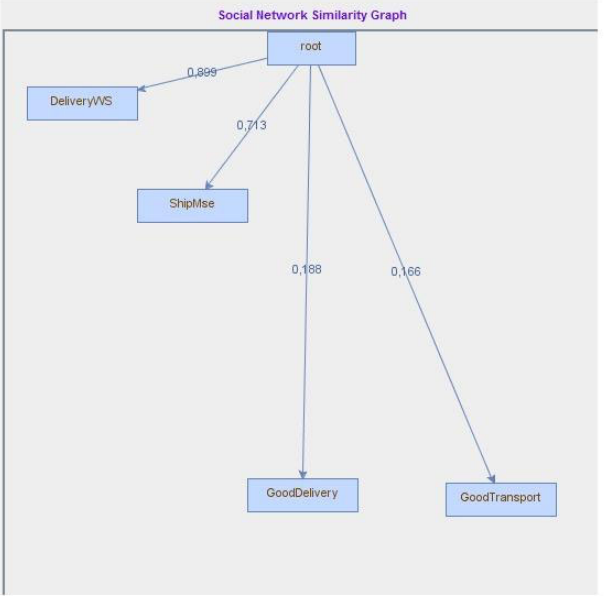


Fig. 6. *ShipperWS*'s substitution social network

5 Conclusion

In this paper we discussed a method to engineer social Web services. This engineering took into account the fact that Web services compete against similar peers during selection, collaborate with different peers during composition, and replace similar peers during failure. These three cases constitute the basis of developing networks of social Web services. The engineering method consists of four steps. In the first step, the objective is to shed the light on the potential

relationships between Web services. In the second step, the objective is to associate these relationships, once identified, with social networks. In the third and four steps, the objectives are to build and analyze these networks so that the social behaviors of Web services are established. Finally, a set of tools like functionality assessment supporting the engineering method were listed. Future work would consist of putting social Web services into action like we did in the *LinkedWS* project, which should permit a further refinement of the method.

References

1. Alrifai, M., Skoutas, D., Risse, T.: Selecting Skyline Services for QoS-based Web Service Composition. In: Proceedings of the the 19th International World Wide Web Conference (WWW 2010), Raleigh, North Carolina, USA (2010)
2. Badr, Y., Maamar, Z.: Can Enterprises Capitalize on Their Social Networks? Cutter IT Journal 22(10) (October 2009)
3. Bengtsson, M., Kock, S.: Coopetition in Business Networks to Cooperate and Compete Simultaneously. Industrial Marketing Management 29(5) (2000)
4. Bui, T., Gacher, A.: Web Services for Negotiation and Bargaining in Electronic Markets: Design Requirements and Implementation Framework. In: Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS 2005), Big Island, Hawaii, USA (2005)
5. Di Martino, B.: Semantic Web Services Discovery based on Structural Ontology Matching. International Journal of Web and Grid Services 5(1) (2009)
6. El-Goarany, K., Saleh, I., Kulczycki, G.: The Social Service Network - Web 2.0 Can Make Semantic Web Services Happen. In: Proceesings of the 10th IEEE Conference on E-Commerce Technology (CEC 2008) and the 5th IEEE Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2008), Washington, DC, USA (2008)
7. Engels, G., Hess, A., Humm, B., Juwig, O., Lohmann, M., Richter, J.P., Voß, M., Willkomm, J.: A Method for Engineering a True Service-Oriented Architecture. In: Proceedings of the Tenth International Conference on Enterprise Information Systems (ICEIS 2008), Barcelona, Spain (2008)
8. Fei-Yue, W., Kathleen, C., Daniel, M.Z., Wenji, M.: Social Computing: From Social Informatics to Social Intelligence. IEEE Intelligent Systems 22(2) (March 2007)
9. Fluegge, M., Santos, I.J.G., Paiva Tizzo, N., Madeira, E.R.M.: Challenges and Techniques on the Road to Dynamically Compose Web Services. In: Proceedings of the 6th International Conference on Web Engineering (ICWE 2006), Palo Alto, California, USA (2006)
10. Foster, H., Uchitel, S., Magee, J., Kramer, J., Hu, M.: Using a Rigorous Approach for Engineering Web Service Compositions: A Case Study. In: Proceedings of the 2005 IEEE International Conference on Services Computing (SCC 2005), Orkando, Florida, USA (2005)
11. Karakostas, B., Zorgios, Y.: Engineering Service Oriented Systems: A Model Driven Approach. IGI Global Publishing (2008)
12. King, I., Li, J., Tong Chan, K.: A Brief Survey of Computational Approaches in Social Computing. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN 2009), Atlanta, Georgia, USA (2009)

13. Maamar, Z., Kouadri Mostéfaoui, S., Yahyaoui, H.: Towards an Agent-based and Context-oriented Approach for Web Services Composition. IEEE Transactions on Knowledge and Data Engineering 17(5) (May 2005)
14. Maamar, Z., Krug Wives, L., Badr, Y., Elnaffar, S., Boukadi, K., Faci, N.: LinkedWS: A Novel Web Services Discovery Model Based on the Metaphor of "Social Networks". In: Simulation Modelling Practice and Theory, vol. 19(10), Elsevier Science Publisher, Amsterdam (2011)
15. Maamar, Z., Sheng, Q.Z., Tata, S., Benslimane, D., Sellami, M.: Towards an Approach to Sustain Web Services High-Availability Using Communities of Web Services. International Journal of Web Information Systems 5(1) (2009)
16. Maamar, Z., Tata, S., Yétongnon, K., Benslimane, D., Thiran, P.: A Goal-based Approach to Engineering Capacity-driven Web Services. The Knowledge Engineering Review Journal, Special issue on Web and Mobile Information Services (2010) (forthcoming)
17. Maaradji, A., Hacid, H., Daigremont, J., Crespi, N.: Towards a Social Network Based Approach for Services Composition. In: Proceedings of the 2010 IEEE International Conference on Communications (ICC 2010), Cap Town, South Africa (2010)
18. Min, L., Weiming, S., Qi, H., Junwei, Y.: A Weighted Ontology-based Semantic Similarity Algorithm for Web Services. Expert Systems with Applications 36(10) (December 2009)
19. Papazoglou, M.: Web Services: Principles and Technology. Prentice Hall, Englewood Cliffs (2007)
20. Xie, X., Du, B., Zhang, Z.: Semantic Service Composition based on Social Network. In: Proceedings of the 17th International World Wide Web Conference (WWW 2008), Beijing, China (2008)

Developing Families of Method-Oriented Architecture

Mohsen Asadi[1,2], Bardia Mohabbati[1,2], Dragan Gašević[2], and Ebrahim Bagheri[2,3]

[1] Simon Fraser University, Canada
[2] Athabasca University, Canada
[3] National Research Council, Canada
{masadi,mohabbati}@sfu.ca, dgasevic@acm.org,
ebagheri@athabasca.ca

Abstract. The method engineering paradigm is motivated by the need for software development methods suitable for specific situations and requirements of organizations in general and projects in particular. Assembly-based method engineering, as one of the prominent approaches in method engineering, creates project-specific methods by (re-)using method components, specified with method processes and products, and stored in method repositories. This paper tries to address the two challenges of assembly-based method engineering related to more effective: i) publication and sharing of method components; and ii) management of variability in software methods, which have many commonalties. In order to address these two challenges, we propose the concept of *Families of Method-Oriented Architectures*. This concept is built on top of the principles of service-oriented architectures and software product lines.

Keywords: Method engineering, Software Product Lines, SOA.

1 Introduction

The increase in the complexity of software-intensive systems has urged for the integration of seminal approaches such as Object-modeling Technique (OMT) and Objectory to form integrated (plan-driven) and unified frameworks such as the Rational Unified Process (RUP). Integrated approaches typically target development of a vast variety of software applications, which increase the size of methods and make them become "cook-book" approaches. Recent critical literature reviews and comprehensive case studies have shown that such cook-book approaches do not work successfully for all circumstances [1]. Practitioners could potentially waste up to 35% of their effort by following the steps of standard development methods [3]. Moreover, the results of such studies reveal that the formal definition prescribed by a method in forms of stages and steps widely differ from the method actually being used [4]. These issues have motivated the software engineering research community to establish the *Method Engineering (ME)* [3] discipline. The ME community concentrates on the idea of providing an "adaptation framework whereby methods are developed to match specific organization situation" [1]. The most prominent ME approach is the *assembly-based method engineering* that creates a new method by assembling

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 168–183, 2011.

existing method components [6][16]. Despite the fact that ME has recently produced promising research results, there are still many open research challenges [1]. In this paper, we focus on two key challenges, namely:

1. The lack of a standard model for describing method components limits the opportunities of method engineers, teams and organizations to share, discover, and retrieve distributed method components. When a method engineer wants to create a new method from scratch or by adapting (extending/constraining) an existing method, a common approach is to try to reuse existing method components from the method repository. Therefore, method components need to be discovered and composed with other method components. Due to the lack of standards, method engineers are forced to reuse method components from the local proprietary repositories, without effective capabilities for retrieving method components in repositories of their collaborators. Moreover, with this limitation of method component sharing, business opportunities of organizations are also limited. In fact, they cannot easily publicize and offer the methods that they are specialized in, as (for profit) services.
2. In essence, organizations initially adopt a method for software development. Afterwards, components of the method may be subsequently added and gradually extended. Such extensions may be derived due to either the evolution of software development or various variations created for some specific method components. Some sources of these diversities may be differences among domains of systems under development (i.e., desktop application, web application, and real-time) or newly emerged software development approaches such as Model Driven Development, Component based Software Development as well as method types such as agile or plan-driven. Thus, there is the need for a systematic approach to manage variability of software methods and adapt software methods (families) that best suit the needs of a specific development context.

The first challenge has been already recognized in the literature [13][1] and some researchers have proposed to use of SOA and Web service standards and principles for dealing with the challenge [13][1]. To this end, the concept of *Method Services* was coined in analogy of the concept of services in SOA. In order to address both of the above challenges at the same time, we propose combining principles (Sect. 4 and 5) of Software Product Line Engineering (SPLE, Sect. 3.1) and Service-oriented Architectures (SOAs, Sect. 3.2) [12]. We use the method service notion for defining method components. We also propose leveraging SPLE with the goal of addressing the second challenge. Our key idea is to introduce a concept of method families, which share a set of common method components, and yet have effective tools for variability management (e.g., feature models). With the use of SPLE principles, we can allow for a systematic modeling of method families and for an automated process of specialization of method families where each family specialization satisfies requirements of a specific situation.

2 Motivating Example

In order to illustrate the challenges that are tackled in this paper, let consider an organization, which develops software systems in two distinct domains, namely,

information systems (both desktop and web-based systems) and *real-time systems*. We consider that the organization has adopted a base method (e.g., RUP) for the entire systems development process. The base method supposedly is a modular method and its method components are stored in a method repository. Moreover, the organization has employed different development approaches, including *code centric development, component based development*, and *model driven development*. Based on the scale and complexity of a project, the organization may follow different development policies such as *agile* or *plan-driven*. In addition, *contingency factors* such as time pressure, user involvement and project familiarity cause the source of diversity in method components. Furthermore, *human factors* (e.g., roles in the organization and their experience level) could be a source of variation points in the method activities. The organization might also intend to add more requirements for future variations of methods and integrate more project management method components in order to have a better support for project management and risk assessment tasks. As a response to the described circumstance, the organization requires to extend the base method using different method components. As a consequence, the complexity and variations of the base method are gradually increased in practice. This complexity leads to a limited sharing and management of lessons learned. Thus, there is a need to more effectively: 1) manage different variations of the base method that were observed and encountered in the previous projects and systematize the lessons learned; 2) anticipate further needs by considering all possible variations of the base method; and create a systematic method for adaptation of the base method considering the needs and requirements of the new development situations. Moreover, the organization, besides its own development projects, might also want to offer some consultancy services or partner with some other organizations based on expertise in method engineering. In such cases, the organization needs to have a standard method for publishing their competencies, so that other organizations can effectively discover and reuse such experience in similar development situations.

3 Background

3.1 Software Product Line Engineering

The SPLE paradigm aims at managing variability and commonality of core software assets of a given domain in order to facilitate the development of software-intensive products and to achieve high reusability [2]. SPLE empowers the derivation of different product family applications (aka, family members) by reusing the realized product family assets such as common models, architecture, and components. In this context, software assets are characterized by a set of *features* shared by each individual product of a family. The set of all valid feature combinations defines a set of product line members of the family. A valid composition of features is called a *configuration* which in turn is a valid software product specialization. The development of a software family is performed by conducting the *domain engineering lifecycle* in which the common assets, family reference architecture and the variability models are developed. Afterwards, in the *application engineering lifecycle*, the common assets

are reused and variability models are configured to produce a family. *Feature modeling,* as a popular technique for modeling variability, is employed to represent the variability and describe the allowed configurations of a software family. This technique is typically used in domain engineering to model an entire family based on the functional characteristics (aka features) that the family provides. Feature models formally and graphically define relations, constraints, and dependencies of software artifacts in a software product family. In essence, there are four types of relationships related to variability concepts in the feature model. They can be classified as: *Mandatory (Required), Optional, Alternative feature group and Or feature group*. Common features among various members of the family are modeled as mandatory features. In other words, mandatory features must be included in the description of their parent features and must be present in any final configuration. Optional features may or may not be included in a final configuration. Alternative features indicate that only one of the features from the feature groups can be opted.

Once a feature model for an entire family is in place, a process of configuration follows. Configuration is a process of selecting features needed for specific applications. Recently, the research community has proposed effective methods for staged configuration where each stage addresses a specific set of requirements in the application development process [11].

3.2 Method Oriented Architecture

Service-oriented computing (SoC) is a computing paradigm that promises flexibility and agility in the development of collaborative software systems. Service-oriented Architecture (SOA) is the main architectural style for realizing the SoC vision. SOA provides an underlying structure enabling for interoperability and communications between services. Web service, reusable and loosely-coupled components, are the best known materialization of SOAs [12]. Web services are built on well-defined standards such as Web Services Description Language (WSDL). Furthermore, the widespread adoption of Web service technologies provides open standards which increase accessibility and interoperability of distributed software services in a networked environment.

On the other hand, ME approaches are hindered by the lack of standards for describing the interfaces of components of methods. Moreover, reusable method components are restrained to be adopted locally by their providers in proprietary repositories. Indeed, the discovery and retrieval of reusable method components can significantly enhance rapid method construction and reuse. The ME community has already proposed the notion of Method Oriented Architecture (MOA) [1][13], which builds on and adopts SOA principles. Rolland proposed the MOA approach where Method as a Service (MaaS) is considered as an analogy to Software as a Services (SaaS)[1]. MOA aims at developing an ME approach, which elevates the accessibility of *method services* and facilitates their automated composition. In MOA, method services are described by method providers through WSDL documents. On the other hand, clients search and retrieve the required method services and compose them in order to create their own more complicated method service.

4 Method Services and Feature Modeling

As mentioned in the introduction section, to address the challenge of variability in method engineering, we intend to apply SPL principles and techniques especially feature modeling. In SPL, functionalities of a set of similar software systems and their visibilities are presented in a feature model in terms of features and variability points, respectively. Likewise, a set of similar methods (we call a family of methods) may have commonalities and variabilities with respect to functionality (i.e. activities). Therefore, a family of methods provides the means for capturing the commonalities (core assets) of all possible methods of a given domain and also addresses variability by covering a comprehensive set of dissimilarities between the methods. In our proposal for family development, distinguishable characteristics of a method mostly including functionalities of the method (i.e. activities) are represented using features. For instance, in our motivating example, one feature of the family is Use-Case modeling. The methods commonality and variability, in terms of their features, are represented in feature model. The development of a family of methods is performed by conducting the *domain engineering* lifecycle (developing feature model and implementing features), which is followed by the *application engineering* lifecycle (developing target method with configuring feature model). We should note that the feature model is only the representation for family characteristics and the variability relations and we need to link them to corresponding method implementations (i.e. method fragment). We use method services as well as MOA techniques (i.e. Method service discovery and composition) to implement features of a family. Therefore, we refer to our approach as development of families of method oriented architectures.

In order to clarify the difference between feature and method service, let us consider the process of method construction as a process of problem solving, in which the requirements model and the final method are considered as the problem space and the solution space, respectively. Since we intended to develop a range (i.e., family) of solutions (i.e. methods) which have common and variable parts, both the problem and solution spaces become more complex. By following SPLE principles, the family problem space (i.e., family requirements model) is decomposed and grouped into features which form a feature model. In other words, a feature intuitively represents sub-problems of the family problem space, and a feature model represents a hierarchical representation of the family space with variability. For instance, the problem space (feature model) of a described family method at the highest level is decomposed into (see Section 6) management, requirement engineering, development, and deployment sub-problems (features). On the other hand, method services form the solution space, in which one or more method services (sub-solutions), implement (solve) one or more features (sub-problems). From another point of view, features address what the properties of the solution are and method services represent the realization of those properties. Fig. 1 shows the *use case modeling* feature (one of the features of the feature model given in Section 6) and the corresponding *use case modeling* method service. As the figure shows, the method service represents how the modeling of use cases should be conducted. Also, a feature represents some functionality, which can be included in a method variant. One of the key concerns in method family engineering is the identification of method services for each feature and the binding of features onto method services. Then, in the application engineering lifecycle, method engineers

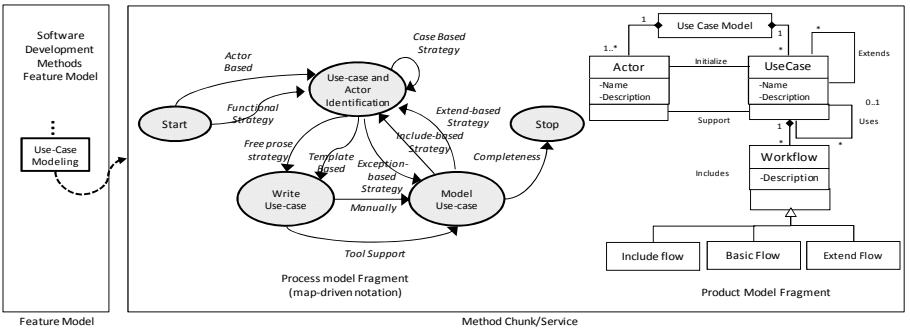


Fig. 1. The relation between use case modeling feature (on the left part) and its corresponding method services which define both process and product model for use-case (adapted from [16])

select features from feature models corresponding to the requirements of the target method (i.e. feature configuration). Next, the method services bound to features in domain engineering are composed automatically and they form an initial method for application engineering. The initial method is adapted and improved until a suitable method is reached for the target problem and deployed.

5 Families of Method Oriented Architecture

Similar to developing software product lines, we propose two main lifecycles for method family engineering process, namely the *Method Domain Engineering* and *Method Application Engineering* lifecycles. Method Domain Engineering lifecycle is carried out one time for the whole family and develops the architecture of the method family, common assets, and variants. In this lifecycle, family features and their variability are modeled by a feature model and suitable method services corresponding to features (i.e. a feature implementation) are discovered and bound to the features. The method application engineering lifecycle develops a target method (i.e. a member of family) for a concrete application by configuring the feature model and assembling the method services related to the configuration. The method application engineering lifecycle is carried out every time a new method is required. The remainder of this section describes the main phases and activities of both lifecycles along with their associated product artifacts.

5.1 Method Domain Engineering

Method domain engineering aims at discovering, organizing, and implementing common assets of a method family. Moreover, determining the scope of a method family and describing the variability of the models is achieved during this lifecycle. The input of the lifecycle is domain knowledge relating to and describing the method family and the reusable assets, while variability models for the methods expressed using feature models are the output of this lifecycle. Fig. 2 illustrates the phases and stages of the method domain engineering lifecycle.

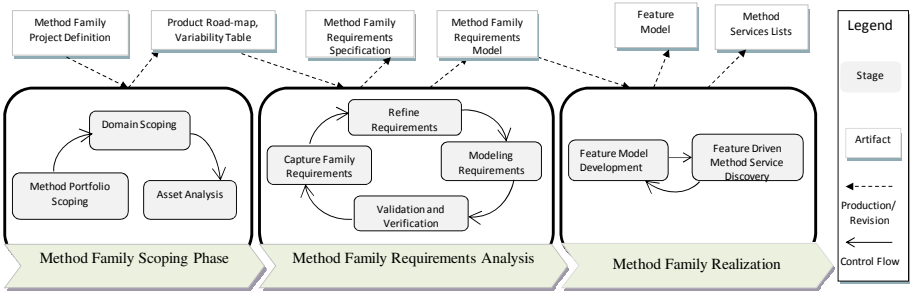


Fig. 2. The Method Domain Engineering Lifecycle

Method domain engineering starts with the *Method Family Scoping* phase which is a key phase for achieving economic benefits of a product line [2]. The Method Family Scoping phase aims at determining a set of products (Software Development Methods) which belong to the family. Scoping of the family is performed in the three stages [2]. The *Method portfolio scoping* stage is a high level domain analysis process and uses the market inputs on existing methods, and expert knowledge to derive a standardized description of a method product line, technical domains that are relevant to it, and the range of methods that shall be supported with the method family. It systematizes the method product information, identifies the main features of the product line and checks the consistency. With regard to features of a method family, the development approaches used in methods (e.g., Model Driven Development-MDD, and SOA), final application domains (e.g., Information System, Embedded Systems, and Ubiquities Systems), and method types (e.g., agile or plan-driven methodologies) are determined through this phase using the project documents.

Later, the *domain scoping* stage uses the basis provided in the previous stage and the expert inputs to identify and group the major functional areas in terms of technical domains which belong to the current method family. Moreover, the benefits and risks pertaining to the various domains are explored and documented. For instance, benefits and risks of employing MDD are identified. Finally, in the *asset analysis* stage, based on the preconditions established in the previous two stages, precise functionality of the method components that should be supported by the method family are described. This stage determines which assets should be developed for reuse (commonality) and which ones as project-specific (variability). The method engineer indicates the variable features (project-specific) belonging to the family, the type of the variables (e.g. logic, workflow), set of variants for the variable, and status of variants (open or close) [17]. The method product-line roadmap is produced as the output of this phase.

The *Method Family Requirements Analysis* phase aims at capturing requirements and developing a requirements model for the methods family. The family requirements model contains unique and unambiguous definitions for each requirement as well as the variability of the requirements. The phase receives the documents, stakeholders' viewpoints, and the product-line roadmap, and variability ranges. Similar to typical software engineering procedures, we define *functional requirements* and *non-functional requirements* for methods. The functional requirements show the properties

that the method should provide, such as work products and required activities; and non-functional requirements include properties that the entire or a large part of methods in the family should have such as smoothness of transition between activities, robustness, and scalability. The method family requirements are *elicited* and *documented*. The method engineer gets an agreement of developers (i.e., stakeholders in this case) on the method family requirements. Next, family requirements are *refined* through *decomposition, aggregation*, and *grouping*. Afterwards, in the modeling family requirements stage, techniques such as the map-driven technique [9] are applied to develop the family requirements model. The family requirements model includes the functional requirements and is represented as family requirements map. The progression activity analyzes the family requirements model and defines the requirements filling the gaps in the family requirements model. Finally, the method engineer verifies the completeness and coherence of the family requirements models as well as the level of satisfaction of the stakeholders' needs by using *Requirements verification* and *Validation* activities [9], respectively.

The goal of the next phase, *Method Family Realization* phase, is to identify common and variant features within the family and to model them with a feature model. Afterwards, the appropriate method services are discovered for each of the features. The feature model is developed by the *Feature Model Development* stage. That is, the common and variable functionalities of methods of the family are managed by representing them in a feature model. The method engineer starts from the requirements and analyzes the requirements, their granularity level and relationships, and then groups them into appropriate features. Moreover, the variability relations are identified between features. Additionally, nonfunctional requirements such as traceability and project management are analyzed and added to the feature model as features and their relations are also identified. Furthermore, the method engineer annotates the features with required information.

The requirements and feature family modeling phases produce the requirements model, requirements documents, and feature model of the method family. Feature family modeling, as described above, is followed by a *Feature Driven Method* for service discovery and selection. The stage of the feature-driven discovery is performed by considering each individual feature and their respective annotations. In essence, a feature annotation provides functional and non-functional keywords used to generate feature queries. The feature queries simply describe what the desired method services should be and how they should behave. In our current implementation, we adopted text-based approach to the discovery of method services. In evaluations of our current implementation of the feature-driven service discovery, we observed promising results in experimenting with the active service search engines while developing families of software services [15]. Since, MOA uses SOA standards for defining and publishing method services, we may expect similar results for discovering of method services. Given that there are no publically available repositories of method services, we are now developing a test collection of method services. In this process, we can easily leverage existing service repositories (e.g., Seekda already used in our implementation) for storing method services. Other approaches can be leveraged in feature-driven service discovery such as logic-based approaches [14].

5.2 Method Application Engineering

Once the method domain model is created, then method engineers can take the method domain model and create different instances out of it based on target method requirements. We refer to this process as *method application engineering*.

Therefore, method application engineering aims to develop a method for a target situation (e.g. a member of the method family) by utilizing the reusable assets created in the domain engineering lifecycle. The input of the lifecycle is the project documents for the concrete method and the output is the method satisfying the requirements. It captures the final method requirements, selects the corresponding features from the feature model, and finally assembles the method services bound to the selected features. The application engineering process is illustrated in Fig. 3.

The *Application Method Requirement Analysis* phase aims to define the requirements of the target method. The documents related to the required method are the inputs and its requirements model and the requirement documents are the outputs.
The documents related to the target method should include *definition* (specify the type of the project at hand), *domain* (specify the application domain of the target method), and *deliverable* (specify the artifacts that should be produced) [10]. The family requirements model and documents are utilized through this phase to produce the method application requirements.

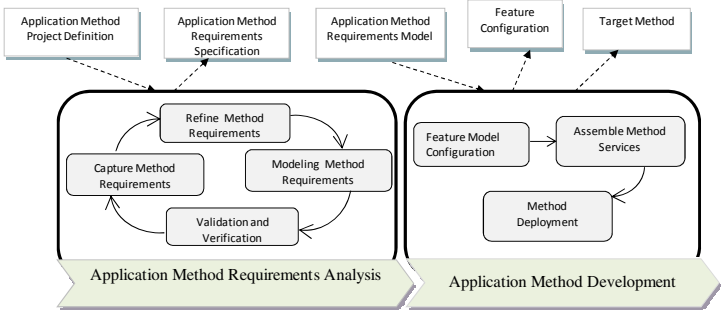


Fig. 3. Method Application Engineering Lifecycle

First, the method application requirements phase captures stakeholders' requirements and documents them. Then, method requirements are refined and clarified further and the agreement of stakeholders is achieved. Next, the method engineer develops the requirements model in the form of a requirements map. Moreover, non-functional requirements are utilized in the feature selection process. Finally, the method requirements are validated and verified to check the completeness and correctness of the method requirements. In all the activities of this phase, the family requirements model is used as a reference to facilitate the process of requirement analysis of the members of the method family. There is a possibility of capturing requirements which were not captured in method family requirement analysis. The

activities of this phase concentrate on one method application, so they do not deal with variability in the family.

The *Application Method Development* phase creates the target method by configuring the method family and delivers the final method configuration to the developers. The method *feature model configuration* stage aims to develop the method by selecting the most appropriate set of features from the feature model through a stage configuration process. It receives the method requirements and produces the corresponding feature configuration. The *stage configuration* process [11] starts from the feature model and carries out successive specializations to create the final configuration. That is, the staged configuration process would limit the space of the method family to the space most relevant for the current method that is being built. Through the staged configuration, the method engineer produces the final configuration. Since in method domain engineering, the method engineer might want to bind a list of method services that have the same interfaces (i.e. situation and intention) but different nonfunctional properties defined in descriptors of method services, the final method service for each feature is selected from the list of alternative method services. The output of the stage is the set of the features (mandatory and optional) as well as their corresponding method services.

If the selected method services (features) do not cover all the requirements of the target method, the new method services for the remaining requirements are discovered in some other repositories or developed from scratch. After the method engineer makes sure that all required method services (features) have been gathered, he/she starts the composition of method services (features) via the *Assemble Method Services* stage. The selected features are divided into functional (e.g., requirement elicitation, use case modeling, and developing design model) and non-functional features (e.g. quality assurance, project monitoring, and traceability checking). First, the method services are orchestrated and the necessary adaptation and mediation are conducted. Then, a decision about the location of method services within a large scope (like quality assurance) is made. After creating the target method, the verification/validation task is done by the method engineer to check whether the method is free from defects and if the target method meets all requirements established in the requirements phase. Moreover, the completeness of the method is verified by a *completeness* task. Finally, the method is deployed to the stakeholder environment by preparing method documents, training developers, and supporting staff through the execution of the method.

6 Case Study

In this section, we represent our motivational example from Section 2 by following our proposed approach described in the previous section. Due to the space limitation, we only explain the domain method engineering lifecycle, which comprises *Product Line Scoping*, *Family Requirement Analysis, Feature Modeling and Feature-based Method Service Discovery and Selection*.

Product Line Scoping: By completing the activities of the product line scoping phase, we identified the criteria which specify the product line boundaries, the main functionality area, and core assets of the method family. Table 1 shows a part of the product line scoping results. One of the major functionality areas distinguished in the domain scoping by all variations of the method is the support for a generic development lifecycle. For instance, unit testing is a core asset in the method family.

Family Requirement Analysis: Functional and non-functional requirements with their commonalities and variability are captured and documented separately. Table 1 shows a part of requirements categorized based on their types. Functional requirements include activities and work products that should be supported with family method. The base method of the organization is explored to discover more detail requirements. The family requirements model is created first by using map-driven approach [6] and then verified and validated. Due to the space limitation for this paper, the requirements model is omitted from the paper.

Feature model Development – based on the family requirements model defined in the previous phase and the existing basis method in the organization, features and their corresponding relations are identified and modeled. The part of feature model designed for target organization is depicted in Fig. 4. Features show the method services required for the family and they can be considered as interfaces for representing method services in the family.

Feature-driven Method Service Discovery and Selection: The next step after feature model development is the discovery of method services. The aim is to find and select among available methods services, which can satisfy desired functional and non-functional requirements of the method for specific situations. As we described earlier, we consider each feature and their associated annotations as queries for method components stored in method repositories. In method service discovery, we assume that the method components, described by WSDL, are available and accessible through either the proprietary method repositories of the organization or public repositories provided by third-parties. Thereby, organizations can publish and share their method chunks as services. Although there are on-line repositories such as Open Process Framework (OPF) [20], available reusable method components are not accessible through standard interfaces. Moreover, there are no facilities to search and discover such available methods. Accordingly, in the process of discovery and selection, the proprietary method repository of the organization is initially used to method services. In case that some of the features are not associated with the organization's services, search queries are broadcasted to the public method repositories.

The Feature Model Plugin (http://gsd.uwaterloo.ca/projects/fmp-plugin/), available for Eclipse environment, is utilized and extended as tool support for modeling and configuring method family. It supports cardinality-based feature modelling, specialization of feature diagrams and configuration based on feature diagrams. Our method chunk service repository is based on the publicly-available Seekda (http://seekda.com) service repository. Our current implementation of feature-driven service discovery is described in [15].

Table 1. Product line scoping and family requirements analysis outcomes after applying the proposed method on the motivational example. It is important to notice that the table does not give all items identified these phased, but some of the most notable examples.

	Phase	Identified work Product and domains
Product Line Scoping	Portfolio	• *Application properties* – application domain (Information systems, Real-time), application type (intra-organization, Organization-customer, inter-organization), source system (it can either use legacy system or does not have system code). • *Development Approach* – systems can be developed by following multiple approaches such as Component based Development, Model Driven Development, or Test Driven Development. • *Human Factors* such skill level includes beginner, medium, and expert (i.e., analyst, designer, developer, and, tester). • *Contingency Factors* –user involvement, project familiarity, project scale and complexity, innovation level, and project dependency. • *Project Management* – monitoring, risk management, configuration and change management, postmortem reviewing, metric management, human resource management.
	Domain	• Generic software development lifecycle (requirement engineering, analysis, design, development, deployment), reusability, management (risk, people), maintenance, test model, implementation models, design model, and Application Technology (Include Data-base, and GUI, is distributed).
	Asset	• Functional requirement engineering, non-functional requirement engineering, behavioral analysis, structural structure, functional analysis, feasibility study, architecture design, project planning, test case development, unit testing, and risk management.
Family Requirement Analysis	Functional Requirements	• *Common* – Specification in high level abstraction, covering generic software development lifecycle, manage and monitor the project, capture requirements, model requirements, validate requirements, defining the infrastructure of system, and plan the project. • Variables- Goal-based requirement extraction, consider review sessions (product and plan review), having stand up meeting, having lightweight design process, formal verification on each abstraction level, concurrency, configuration of software and hardware, having platform independent models, having platform specific models, component identification, component specification, component interaction, component assembly, and PIM and PSM synchronization.
	Functional Requirements	• Common – iterative process, incrementally development, traceability to requirements, clear separation of concerns, smooth transition between activities, and method flexibility. • Variables - semi automatic refinement between abstraction level, method scalability, lightweight process, and formal checking.

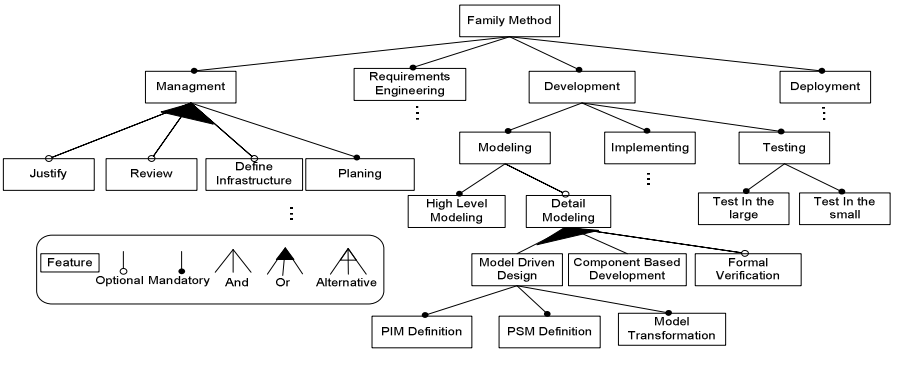


Fig. 4. A sample feature model of a family of software development methods

7 Related Work

ME defines techniques and approaches for constructing and/or adapting the methods. The most prominent sub-area of ME, *Situational Method Engineering* (SME), proposed by Welke et al [5] is concerned with the creation of methods 'on-the-fly' (i.e. construct or adapt a method according to situation of the project at hand). The ME approaches are classified by Ralyte et al [6] as: *Ad-Hoc* (i.e. Method created from scratch); *Extension-Based* (i.e. Method is created by extending an existing method [6]); *Paradigm-based* (an existing meta-model is adapted, instantiated, or abstracted to create a new method [6]); and *Assembly-based* (a method is created by reusing existing method components [7][16][25]). These approaches mostly focus on reusability and modularity principles. Besides this classification, Karlsson et al. [8] proposed the *Method Configuration* approach (more general than extension based) in which a target method is created by adding/removing elements and features. They concentrate on variability management and reusability. All mentioned approaches are based on one or more of the following principles - meta-modeling, reuse, modularity, and flexibility. Our proposed approach is similar to the assembly-based and method configuration by following of the modularity, reusability, and variability principles. However, our approach enables for a higher degree of reusability by leveraging SOA principles and for a more systematic variability management by employing SPLE principles (As shown in software engineering SPLE increases reusability [24]).

Gonzalez-Perez [20] explained the benefits of ISO/IEC 24744 meta-model for both method specification and enactment and proposed a product-centric approach to developing a new methodology. Aharoni et al [22] enriched the ISO/IEC 24744 for creating and tailoring methods through an approach called Application-based Domain Modeling (ADOM). The approach is based on the layered framework including application, domain, and language. The domain (methodology) layer contains different method concepts as well as the specification of their exact usage situation. The application layer, called endeavor layer, includes specific method components and situational methods, which are created based on domain model terminology, rules, and constraints. The language layer defines any modeling language that can be used for describing meta-models and method components. Our approach differs from these approaches in using variability modeling language (i.e. feature modeling) and software product line principles. Moreover, we provide a reference architecture for a whole family which eases configuring and developing methods. Additionally, we use a new concept for method component (i.e., method service), which utilizes standards in SOA to improve discovering and reusing method components.

Recently MOA [13][1] was proposed which empowered the assembly-based method engineering principles with a standard for describing method components (in terms of method service) and with service discovery principles for finding distributed method components. Our approach also utilizes MOA to describe and discover method services corresponding to the features of a method family.

8 Discussion

Two main issues regarding the proposed approach are validity and cost-benefit analysis of the approach. For both issues, it is required to conduct an empirical study. We

did a case-study in which we explained the steps of the method. However, it cannot completely ensure the validity of our approach, its benefits and limitations. In order to clarify these issues in our method, we make our argument based on analogy between software and methods as proposed by Osterweil [23]. Therefore, our assumption is "software processes are software too" [23]. Considering this analogy, we can adopt similar approaches and techniques used in software engineering for solving existing problems in method engineering. As we see, the method engineering community proposed MOA inspired from SOA to deal with the lack of standard for defining the method fragment interfaces [1]. But, to use analogy as a viable strategy for solving a problem in method engineering, referred to as the target domain, we need to identify the corresponding construct in software engineering (source domain) and define a mapping schema. For example, in method engineering, the method fragment notion (called method service) is mapped to service notion in SOA and method notion is mapped to software service. Hence, we can use SOA principles and have benefits of SOA in the MOA domain. The other problem method engineers deal with is variability in the base method and configuring the method based on the target project for which some approaches have been proposed [6][8]. On the other hand, software variability is a well-known problem in the software engineering community and many techniques and approaches have been proposed like feature modeling to manage the problem and various success stories in using product families and associated techniques have been reported. As an example, Clements and Northrop reported that Nokia was able to increase its production capacity for new cellular phone models from 5-10 to around 30 models per year, which alleviated Nokia's main challenge being the high pace of market demand and customer taste change [24]. These results ensure both validity and benefits of software families. Therefore, we tried to make an analogy between software family and method base and coined the notion of family of methods. We mapped the features to the method fragment interfaces and handled the variability in base method and configuration problem according to the target project requirements. As a result, we expect similar benefits to be reaped within the method engineering domain. We are also aware of the cost of creating family or reengineering current methods into method family (i.e., creating a method feature model), but for long term the benefits that will be achieved can recompensate these costs as happened in the broader software engineering practice.

9 Conclusion and Future Work

In this paper, we have presented an approach for developing families of software developments methods. We exploited the notion of method services to facilitate the discovery of distributed method components. Such discovered method components can be used as an implementation for both sets of common and variable method assets of a family of methods. The proposed approach makes use of feature modeling to manage variability of method families. Managing and modeling variability enables for a more effective method construction and for a more systematic method reuse. We believe that the described concept of families of method-oriented architectures may not be entirely feasible now, due to the lack of a complete method sharing ecosystem, but with the growing interest for services economy, more attention to such ecosystems

can easily be envisioned to appear soon. Thus, our approach is a small step towards making this vision possible. The adoption of widely used SOA standards helps in publishing and sharing method components. Furthermore, organizations will be able to take the advantages of distributed architectures to design, implement, execute and reuse available method components. Last but not least, the long term goal is to enable different organizations and enterprises to publish, advertise, discover and reuse their methods components.

While the paper proposed a methodology for the combined use of SPLE and SOA principles in method engineering, the contribution of this paper deserves to be considered in a broader context of its implications. As already demonstrated in the previous research [18], transforming configured feature models into workflow and service composition languages is possible. Thus, the combined use of SOA and SPLE enables for leveraging existing workflow engines (e.g., BPEL) in management and execution of software projects. Moreover, with such an executable representation of methods as workflows, one can also expect an increased compliance of projects with the steps defined by methods. As workflow management provides also best practices (i.e., workflow patterns), the combined use of workflows with software methodologies might lead to further benefits such as improved parallelization of some stages. With representation of method components as services, tracking of the project progress could also be improved, while the invocation of method services can explicitly be associated with the other tools used in different method stages.

As future work, we intend to provide a more comprehensive evaluation of the proposed approach by developing a collection of method service to be used for experimentation. We aim to extend our approach from different perspectives to reduce the manual intervention needed in the final method development. We plan to use ontology-based representation for feature models to automate consistency checking of features in method families as described in [18]. Furthermore, we intend to extend the feature modeling language to allow method engineers to add concepts of domain ontologies for annotation of features. This will consequently be used for advanced ontology-based discovery and composition of method services [14]. Currently, we are developing an environment that supports our proposed process. The environment will include the modeling of method families, annotation of feature models, discovering of method services, stage configuration of feature models, and deployment to standard workflow management engines.

References

1. Rolland, C.: Method engineering: towards methods as services Software Process. Improvement and Practice 14(3), 143–164 (2009)
2. Schmid, K.: A comprehensive product line scoping approach and its validation. In: Proc. of the 24th International Conference on Software Engineering, pp. 593–603 (2002)
3. Harmsen, A.F.: Situational Method Engineering. Moret Ernst & Young, Utrecht (1997)
4. Lings, B., Lundell, B.: Method-in-action and method-in-tool: some implications for case. In: Proc. 6th Int'l Conference on Enterprise Information Systems, pp. 623–628 (2004)
5. Welke, R.J., Kumar, K.: Method Engineering: a proposal for situation-specific methodology construction. In: Cotterman, W.W., Senn, J.A. (eds.), pp. 257–268. Wiley, Chichester (1992)

6. Ralyté, J., Deneckére, R., Rolland, C.: Towards a generic model for situational method engineering. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 95–110. Springer, Heidelberg (2003)
7. Ralyté, J., Rolland, C.: An assembly process model for method engineering. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 267–283. Springer, Heidelberg (2001)
8. Karlsson, F., Gerfalk, P.J.A.: Method configuration: adapting to situational characteristics while creating reusable assets. Inf. and Soft. Technology. 46(9), 619–633 (2004)
9. Ralyte, J.: Requirements definition for the situational method engineering. In: Proc. of the IFIP WG8.1 Working Conf. on Eng. Inf. Sys. in the Internet Context, pp. 127–152 (2002)
10. Coulin, C., Zowghi, D., Sahraoui, A.E.K.: A Lightweight Workshop-Centric Situational Approach for the Early Stages of Requirements Elicitation in Software Systems Developme. In: Proc. of Workshop on Situational Requirements Eng. Processes (2005)
11. Czarnecki, K., et al.: Staged Configuration through Specialization and Multi-level Configuration of Feature Models. Soft. Process: Improvement & Prac. 10(2), 143–169 (2005)
12. Tsai, W.: Service-oriented system engineering: a new paradigm. Service-Oriented System Engineering. In: Proc. IEEE Int'l Workshop on Service-Oriented Sys. Eng., pp. 3–6 (2005)
13. Deneckère, R., Iacovelli, A., Kornyshova, E., Souveyet, C.: From Method Fragments to Method Services. In: Proc. 13th Int'l Conf. on Exploring Modelling Methods for Systems Analysis and Design, pp. 81–96 (2008)
14. Klusch, M.: Semantic Web Service Coordination. In: CASCOM: Intelligent Service Coordination in the Semantic Web, pp. 59–104 (2008)
15. Mohabbati, B., Kaviani, N., Lea, R., Gašević, D., Hatala, M., Blackstock, M.: ReCoIn: A Framework for Dynamic Integration of Remote Services in a Service-Oriented Component Model. In: Proceedings of the 2009 IEEE Asia-Pacific Services Comp. Conf. (2009)
16. Mirbel, I., Ralyte, J.: Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering 11(1), 58–78 (2006)
17. Kim, S., Min, H.G., Her, J.S., Chang, S.H.: DREAM: A practical product line engineering using model driven architecture. In: Proc. Int'l Conf. on Information Technology and Applications, pp. 70–75 (2005)
18. Montero, I., Pena, J., Ruiz-Cortes, A.: From Feature Models to Business Processes. In: Proc. of the IEEE Int'l Conf. on Services Computing, vol. 2, pp. 605–608 (2008)
19. Bošković, et al.: Automated Staged Configuration with Semantic Web Technologies. International Journal of Software Engineering and Knowledge Engineering (in press, 2010)
20. OPEN Process Framework (OPF) Web Site, `http://www.opfro.org/`
21. Gonzalez-Perez, C.: Supporting Situational Method Engineering with ISO/IEC 24744 and the Work Product Pool Approach. SME: Fundamentals and Experiences, pp. 7-18 (2007)
22. Aharoni, A., Reinhartz-Berger, I.: A Domain Engineering Approach for Situational Method Engineering. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 455–468. Springer, Heidelberg (2008)
23. Osterweil, L.: Software processes are software too. In: Proceedings of the ICSE, pp. 2–13. IEEE Computer Society Press, Monterey (1987)
24. Clements, P., Northrop, L.M.: Software product lines visited June 2010 (2003), `http://www.sei.cmu.edu/programs/pls/sw-product-lines0503.pdf`
25. Asadi, M., Ramsin, R.: Patterns of Situational Method Engineering. In: Lee, R., Ishii, N. (eds.) Software Engineering Research, Management and Applications (SERA) 2009. SCI, vol. 253, pp. 277–291. Springer, Heidelberg (2009)

Agile Service Development: A Rule-Based Method Engineering Approach*

Stijn Hoppenbrouwers[1], Martijn Zoet[2], Johan Versendaal[2,3], and Inge van de Weerd[3]

[1] Radboud University Nijmegen, The Netherlands
s.hoppenbrouwers@cs.ru.nl
[2] University of Applied Sciences, Utrecht, The Netherlands
{martijn.zoet,johan.versendaal}@hu.nl
[3] Utrecht University, Utrecht, The Netherlands
{j.versendaal,i.vandeweerd}@cs.uu.nl

Abstract. Agile software development has evolved into an increasingly mature software development approach and has been applied successfully in many software vendors' development departments. In this position paper, we address the broader agile *service* development. Based on method engineering principles we define a framework that conceptualizes an operational way of working for the development of services, emphatically taking into account agility. As a first level of agility, the framework contains situational project factors that influence the choice of method fragments; secondly, increased agility is proposed by describing and operationalizing these method fragments not as imperative steps or activities, but instead by means of sets of minimally specified, declarative rules that determine the context and constraints within which goals are to be reached. This approach borrows concepts from rules management, organizational patterns, and game design theory.

Keywords: method engineering, agile service development, business rules, business rules management, product management, game design.

1 Introduction

To remain competitive, organizations are increasingly urged to adapt to changes in their business environment. Trends like higher demanding customers, faster changing customers' demands, increased regulation, and offshoring give rise to the re-thinking of business models and processes. In the software development industry, a number of vendors have successfully applied 'agile software development process' principles,

* This paper results from the Agile Service Development project (*http://www.novay.nl/okb/projects/agile-service-development/7628*), a collaborative research initiative focused on methods, techniques and tools for the agile development of business services. The project consortium consists of BeInformed, BiZZdesign, CRP Henri Tudor, HU University of Applied Sciences Utrecht, IBM, Novay, O&i, PGGM, RuleManagement Group, Radboud University Nijmegen, Twente University, Utrecht University, and Voogd & Voogd. The project is part of the program Service Innovation & ICT of the Dutch Ministry of Economic Affairs.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 184–189, 2011.

decreasing time-to-market and addressing rapidly changing customer demands. Approaching this more generically, any business may likewise apply concepts of agility as a strategy to take up the described challenges in the business environment. Agility is defined as "the ability of a sensitive [organization] that exhibits flexibility to accommodate expected or unexpected changes rapidly, following the shortest time span, using economical, simple and quality instruments in a dynamic environment and applying updated prior knowledge and experience to learn from the internal and external environment" [1]. The aforementioned definition positioned in the context of agile service development asserts that an organization should be able to create or adapt a (business) service efficiently and effectively when changes occur in its environment. A business service is considered an externally visible and accessible unit of functionality offered by an organization to its environment, delivering a meaningful value to that environment. An example of such a service is 'an insurance product tailored towards singles'.

Agile development is not an alien concept in management and information systems research. It plays some role in existing work on *situational method engineering* in software product development literature [2, 3, 4, 5]. These studies acknowledge the need for development methods tuned to the situation of the project at hand. Based on situational factors distilled from the project, meta-methods composed of outlines or more detailed procedures, are selected and integrated into a coherent method appropriate for that specific situation [4].

However, 'situational' is not synonymous to 'agile'. For a method to become truly agile, changing situational factors also have to be linked (if required) to 'run time', changes in the method: quick responses to new situational information, and the installation of short feedback loops applying to the method. Existing studies mainly focus on situational fit of the overall development process while still describing the actual method fragments in terms of 'non-agile', step-by-step, instructions inherent to traditional workflow-like process descriptions.

2 Method Engineering for Agile Service Development

Situationality is the ability of a method to respond and adapt to a specific environment based on defined characteristics [4, 6]. Although scholars approach the concept from different viewpoints, the fundamental basis is the creation of reusable method parts called method fragments [7] or method chunks [8]. The method fragments are stored in a repository called the method base. In addition to the method fragments, also assembly rules and situational factors are stored inside the method base [5].

Utilizing the perspective of situationality, method fragments can be used to provide some degree of agility with respect to the project at hand. Regarding the assembly of method fragments, our approach follows the configuration process for situational method engineering as proposed by Brinkkemper [4]. However, our approach adds a second dimension of agility in operational execution.

Due to changes, predictable or unpredictable, in the environment, the method must be able to quickly adjust to the situation at hand. The method engineering process proposed by Brinkkemper [4] incorporates this by means of a build-in feedback loop. This feedback loop facilitates selecting new process alternatives in terms of method

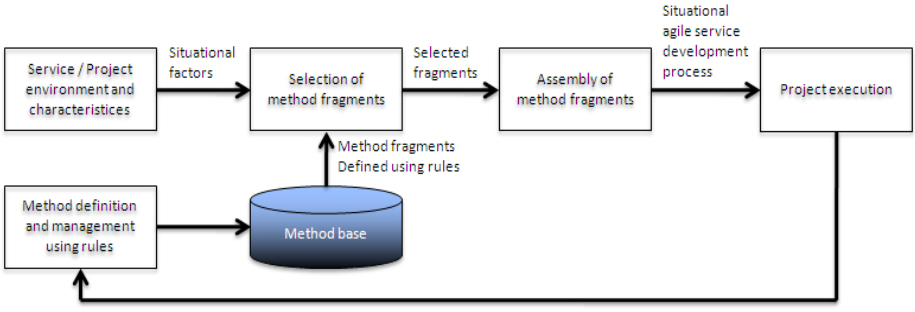


Fig. 1. Method engineering approach for agile service development

fragments hereby inserting the underlying assumption that changes in the environment will result in replacing complete method fragments. We argue that changes in the environment will not always lead to changes in the executed method but can still influence the operational execution of a specific method fragment.

To realize this, we propose a particular operationalization of the method engineering approach and process in terms of the selection process of method fragments, situational factors and assembly rules. The idea is that participants are given as much freedom as possible within necessary methodical and contextual constraints (minimal specification), and that the ability to respond quickly to desired changes in the method (as indicated by fast feedback) is optimized: increased agility in our approach is supported by defining method fragments in a rule-based, declarative manner. This approach is inspired by principles and practices from (business) rules management, organizational patterns and game design theory. Rule-based specification of methods is vaguely suggested in [9], who argue in favor of using practices instead of processes in software engineering. Our approach is inspired by this line of thinking, but pushes for advanced description, management and operationalization of 'method rules' in a specific service development context.

In the following subsections the method engineering meta-model by Brinkkemper [4] will be described in some more detail for 1) situational project factors and characteristics, and 2) method fragment description and identification (see figure 1).

2.1 Situational Project Factors and Characterization

Situational factors can be used to characterize projects, processes, and companies. Bekkers [10] researched the influence of situational factors on the practice of software product management, which resulted in a list of 27 situational factors, divided over the categories (1) business unit characteristics, (2) customer characteristics, (3) market characteristics, (4) product characteristics, and (5) stakeholder involvement. A situational factor influence is, for example: "the amount of requirements that are submitted by the customers has a high impact on how requirements management processes should be carried out". If this situational factor were to change, the company should also change its processes in order to cope with this change. We intend to apply the 27 situational factors in the context of agile service development.

2.2 Method Fragments Description and Identification

We choose a rule-based, declarative approach to the description of method fragments. Declarative description allows for minimal specification. In an agile environment, 'just enough' explicit regulation of the way of working is to be preferred over imperative style, step-by-step instruction inherent to traditional flow-like process description. The declarative approach is mainly what has led us to introduce the 'game metaphor' as an image of how we intend to deal with describing agile methods and method fragments.

As suggested in Hoppenbrouwers [11, 12], methods can be fruitfully viewed as games. They have clear objectives and rules the participants are to comply to, or at least choose to be guided by. The driving concepts in this approach are *goals*. These can cover all aspects of what one wants to achieve (deliverable or product goals, like 'create an insurance product for singles, within 2 months') and how one wants to do this (process goals, like 'use SCRUM'; 'comply to HIPAA regulation'; 'actively involve representatives of prospective customers'). Many kinds of goals can be distinguished, and all of them can be represented in the form of rules. Goals are thus covered by *goal rules*.

To realize goals, activities are needed. If goals are logically ordered, so are the activities linked to them (like 'hold SCRUM standup meeting'), which can be planned in space and time, allocating specific people and resources. Activities can be temporally ordered, but do not need to be in principle. This is in line with the principles of declarative workflow [13] and allows for minimal specification: formally planning only what needs to be planned, and leaving the rest to the team's powers of self-organization.

Not only goals can be expressed as rules, but also the temporal ordering (*procedural rules*: *x* before *y*) and even constraints on interaction: *interaction rules* that concern who talks to who ('tester *t* with stakeholder *s*') and by what means ('using think-aloud session using prototype PT2.1'). This links high-level method engineering to more operational method engineering involving communication situations [14]. Additional rules can cover aspects like the format or language (i.e. meta-model notation: 'UML Use Cases, Class Diagrams, Activity Diagrams') of any deliverables strived for. At the operational level of communication situations, the rules have to be specific and readable enough to effectively guide people in their activities –in as far as such guidance is required (minimal specification).

There is a clear parallel between a declarative, rule-based approach, the game metaphor, and the use of patterns; in particular, organizational patterns [15]. Cockburn has advocated game-theoretical use of the game metaphor in studying the software engineering process [16], but not in the applied sense we now propose. Our rules for describing method fragments will cover principles and patterns of agile practice (including many existing ones), and operational reflections thereof.

3 Conclusions

In view of increasing demands for agility in processes for service development, we are in the early stages of applying existing principles and practices from situational

method engineering to service development processes and methods, combining these with approaches supporting agile process management and execution. On the method engineering side, this requires some innovation concerning the description, management, and operationalization of methods. Without claiming that the approach put forward in this position paper will guarantee agility of processes for service development, we believe the approach proposed will allow for considerably better agility than existing practices in ME that are more rooted in imperative style specification of methods and method fragments. Our rule-based approach should enable quick adaptation of the method's 'rules of the game' to changing situational factors. 'Games played' will be short cycles or phases in development, in line with widespread agile practices in software engineering. In addition, we pay explicit attention to operationalization of methods by specifying actual 'games to be played' in terms of concrete 'communication situations', and linking these to higher level goals and activities as included in some method and drawn from the method base.

We will test and refine our approach to method engineering in agile service development in close cooperation with a number of partners from industry. We will explore our approach in the re-engineering of past project cases, but will also, even in the early stages of investigation, start applying our framework in real cases of running projects.

Our approach can be seen as complementary to another innovative direction in Method Engineering: that of 'Method as a Serivce' (MaaS) [17]. Method fragments are developed as *method services* which are implemented as web services. To make the method services widely available, a Method-Oriented Architecture (MOA) is proposed. With the concept of MaaS, the authors aim to overcome many drawbacks that exist with existing method fragments, such as lack of interoperability, and lack of interactivity.

References

1. Qumer, A., Henderson-Sellers, B.: An evaluation of the degree of agility in six agile methods and its applicability for method engineering. Information and Software Technology 50(4), 280–295 (2007)
2. Olle, T.W., Hagelstein, J., MacDonald, I.G., Rolland, C., Sol, H.G., van Assche, F.J.M., Verrijn-Stuart, A.A.: Information Systems Methodologies: a Framework for Understanding, 2nd edn. Addison-Wesley, Reading (1991)
3. Kumar, K., Welke, R.J.: Methodology engineering: a proposal for situation-specific methodology construction. In: Cotterman, W.W., Senn, J.A. (eds.) Challenges and Strategies for Research in Systems Development (1992)
4. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information and Software Technology 38(4), 275–280 (1996)
5. van de Weerd, I., Versendaal, J., Brinkkemper, S.: A product software knowledge infrastructure for situational capability maturation: Vision and case studies in product management. In: Proceedings of the 12th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2006), Luxembourg, pp. 97–112 (2006)
6. Ralyté, J., Deneckère, R., Rolland, C.: Towards a generic model for situational method engineering. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, p. 95. Springer, Heidelberg (2003)

7. Harmsen, F., Brinkkemper, S., Oei, H.: Situational method engineering for information system project approaches. In: Verrijn Stuart, A.A., Olle, T.W. (eds.) Methods and Associated Tools For the Information Systems Life Cycle, Proceedings of the IFZP WG8.1 Working Conference CRIS 1994, Maastricht, pp. 169–194. North-Holland, Amsterdam (1994)
8. Rolland, C., Plihon, V., Ralyté, J.: Specifying the reuse context of scenario method chunks. In: Pernici, B., Thanos, C. (eds.) CAiSE 1998. LNCS, vol. 1413, p. 191. Springer, Heidelberg (1998)
9. Jacobson, I., Ng, P.W., Spence, I.: Enough of Processes - Lets do Practices. Journal of Object Technology 6(6), 41–66 (2007), http://www.jot.fm
10. Bekkers, W., van de Weerd, I., Brinkkemper, S., Mahieu, A.: The influence of situational factors in software product management: an empirical study. In: Proceedings of the 2nd International Workshop on Software Product Management, Barcelona, Spain, pp. 41–48 (2008)
11. Hoppenbrouwers, S.J.B.A., Weigand, H., Rouwette, E.A.J.A.: Setting Rules of Play for Collaborative Modelling. Kock, N., Rittgen, P. (eds.) International Journal of e-Collaboration (IJeC) 5(4), 37–52 (2009)
12. Hoppenbrouwers, S.J.B.A., van Bommel, P., Järvinen, A.: Method Engineering as Game Design: an Emerging HCI Perspective on Methods and CASE Tools. In: Proceedings of EMMSAD 2008 (Exploring Modelling Methods for System Analysis and Design), held in conjunction with CAiSE 2008, Montpellier, France (June 2008)
13. van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science-Research and Development 23(2), 99–113 (2009)
14. Hoppenbrouwers, S.J.B.A., Wilmont, I.: Focused Conceptualisation: Framing Questioning and Answering in Model-Oriented Dialogue Games. In: van Bommel, P., Hoppenbrouwers, S.J.B.A., Overbeek, S., Proper, H.A., Barjis, J. (eds.) PoEM 2010. Lecture Notes in Business Information Processing, vol. 68, pp. 190–204. Springer, Heidelberg (2010)
15. Coplien, J., Harrison, N.: Patterns of Agile Software Development. Addison-Wesley, Reading (2004)
16. Cockburn, A. (2004). The End of Software Engineering and the Start of Economic-Cooperative Gaming (2004), http://alistair.cockburn.us/The+end+of+software+engineering+and+the+start+of+economic-cooperative+gaming (retrieved October 23, 2010)
17. Rolland, C.: Method Engineering: Towards Methods as Services. Software Process: Improvement And Practice 14(3), 143–164 (2009)

Bridging the Gap between Business Processes and Service Composition through Service Choreographies

Mario Cortes Cornax, Sophie Dupuy-Chessa, and Dominique Rieu

University of Grenoble, CNRS, LIG
SIGMA Team, BP 72, 38402 Saint Martin d'Heres, Cedex France
{Mario.Cortes-Cornax,Sophie.Dupuy,Dominique.Rieu}@imag.fr
http://sigma.imag.fr/

Abstract. Inter-organizational business processes implementations using service composition approaches are being more and more used. We want to reduce the semantic gap that exists between both worlds (business processes and services) through service choreographies, a composition approach that we think it is semantically close to multi-party business processes. We rely on modeling techniques as abstraction layers and view separation to achieve our goal. Our start point is a web service choreography meta-model presented in three abstraction layers where each layer is divided in a structural and a behavioral view. The meta-model can be used in a top-down or a bottom-up approach to make a progressive transition between the business process and the service world.

Keywords: Choreography, Business Processes, Modeling Techniques.

1 Introduction

Today, organizations are moving towards inter-organizational business processes. Therefore, they depend on other organizations. To model their business processes, analysts and designers use graphical languages that allow an intuitive and easy reading such as Business Process Management Notation (BPMN) [11]. Modular solutions based on service composition [16] are increasingly found to implement business processes. Service composition languages are mainly based on XML and they do not have a graphical standard notation. We observe a semantic gap between the way of describing business processes and the way of implementing them with service composition, what can create ambiguities and unexpected results as described in [14].

A big effort has been done to bring BPMN closer to web services. In [1] Wil M.P. van der Aalst et al. survey several papers issue of the interest to bridge the gap between these two worlds. BPMN and BPEL [10] alignment is exposed in the standard BPMN. However, we observe that mapping efforts are centered in orchestrations i.e. in two by two relationships between the different external entities of a single process. As business process complexity increases and depends

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 190–203, 2011.

on other organizations, this pair relationships remains insufficient to manage the complexity when several organizations share common goals. This constraint can negatively influence a global understanding of the process limiting its potential capacity of optimization [14]. A global viewpoint besides a set of individual viewpoints is required to better understand, build, survey and optimize the global process.

In this context, an interesting concept is the *web service choreography* [6]. Web service choreography is a service composition approach focused on message exchanges between different services from a global viewpoint. It can be understood as a multi-party communication protocol where there is no central coordinator. This composition approach seems to be close to inter-organizational business processes. We will therefore seek to better understand service choreographies to bridge the gap between the two worlds that are business processes and service composition when working in multi-party scenarios.

We rely on modeling techniques to propose a meta-model that aims at defining the semantics of choreographies. The meta-model reduces ambiguity and clarifies the main elements of a language. Our approach is based on the concepts of *views* and *abstraction levels*. We remark the necessity of several abstraction layers to provide adapted viewpoints to the different actors that contribute to set up an inter-organizational business process that relies on SOA. Providing these abstraction levels and defining progressive transitions between them is the way to achieve the approach between the global process design and the executable processes. In [18] authors argue the necessity of three abstraction layers when vertical model alignment is targeted.

The rest of paper is organized as follows. Section 2 describes the meta-model approach and compare our proposal with related works. The meta-model overview based on a scenario and a brief description of each layer are presented in Section 3. Section 4 presents a discussion that summarizes our work and future perspectives.

2 Overview of the Approach

Our meta-model construction focuses on three main axis illustrated with arrows in Fig. 1. It starts in a deep analysis of the Web Service Choreography Description Language (WS-CDL) [17] that brings us the knowledge about choreography to build a design meta-model. WS-CDL has been criticized in [3] by Barros et al. as it does not have a clear semantic model and it presents a difficult alignment with BPEL[10]. The Web Services Choreography Working Group [19] stopped the development of this language in July 2009 but it is still a reference as choreography language. There has been other language proposals, mostly developed in research projects. In [8] Decker et al. introduce BPEL4Chor [7] and survey some other choreography language proposals as Let's Dance [20] or iBPMN [5], an extension of BPMN adapted to choreographies. But this concept is still far to be standardized and adopted by the industry. It seems that all proposals are converging to the new choreography representation presented by the OMG in

it's new BPMN version 2.0 [12], but the future of choreography still remains uncertain.

Our design meta-model is based on the one presented by Barros et al. in [3] but enriched by dividing it in different packages. The analysis meta-model is an abstraction of the design meta-model where the syntax dependencies of the specific language were removed. The domain meta-model is an abstraction of the latter one where the fundamental choreography concepts are represented. One of our next objectives is to present in the lowest abstraction layer a language meta-model that is closer than WS-CDL to executable process like BPEL4Chor. Choreography language proposals like BPEL4Chor or the Multi Agent Protocol (MAP) language presented in [2] are well aligned to final executable processes but they lack of a more abstraction viewpoint to reach the business level. We find the necessity to present the choreography concept in three traditional abstraction layers (domain, analysis and design) to bring closer both worlds.

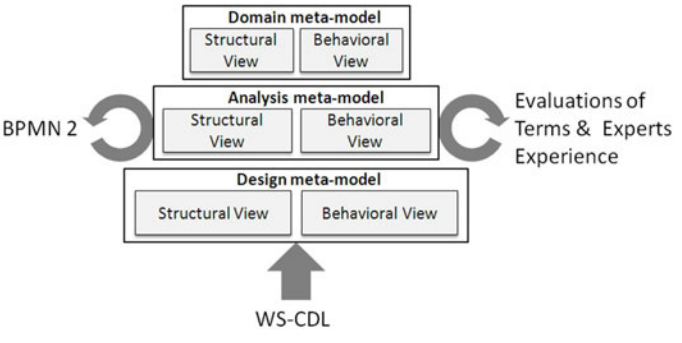


Fig. 1. The construction approach of the meta-model

We observe in Fig 1 that settling the analysis meta-model has been our main target until today. Several iterations at this level were done. An empiric evaluation was performed where business and technical experts examined the suitability of the terms as well as relationships between them through a scenario. We also surveyed some ideas from the choreography diagrams of BPMN's new version [12]. As result, we obtained a simple but representative implementation-independent choreography meta-model. As it is defined in [6] an *implementation-independent level* is where "fundamental decisions about interactions are made", avoiding concrete message formats or security issues. In future works we envisaged to validate in a more formal way our approach against the *Service Interaction Patterns* [4] which are the better known benchmark when working in multi-party collaborative environments. We would like to introduce pattern concepts in our meta-model.

Fig. 1 also shows a division in two different points of view of each layer: the structural and the behavioral views. This separation makes the models easier to read and more understandable for both designers and developers. The structural view connects all fundamental components displaying them in a static way. The

behavioral view defines the reactions of the different elements to the actions of the others. When analyzing WS-CDL, we identified the elements corresponding either to one view or the other. We represented this separation from the first layer (the design meta-model) and we maintained it, in the consecutive abstractions. In this paper we focus on the meta-model and the modeling techniques.

3 The Choreography Meta-model

This section presents the choreography meta-model layers. We use a scenario describing a computer purchase by a customer and manufacturer's delivery protocol. We will therefore present our meta-model following the scenario taken from [12].

3.1 Scenario

John is a *customer* that orders a custom computer to an Internet's *manufacturer* called ComputerSeller.com. ComputerSeller.com is part of a computer manufacturer's network which they have established a protocol to deliver a purchase order in an efficient and speedy way:

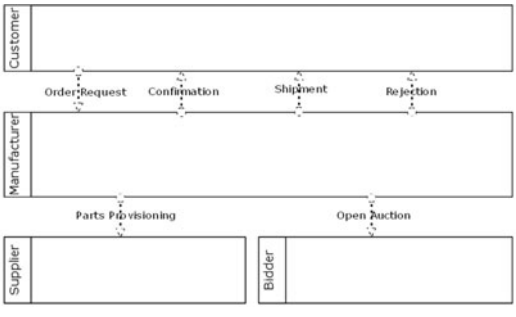


Fig. 2. Scenario interactions represented through a BPMN's collaboration diagram

- If the manufacturer can provide the order, it sends a confirmation and then performs the delivery.
- If the manufacturer can not respond to this command, it rejects the order, explaining the denial reasons.
- If a manufacturer can satisfy the order but it does not have all the pieces available, it should contact a *supplier* so that it provides the missing pieces. The supplier could then procure the missing pieces.
- After contacting the supplier, if the manufacturer has all the parts available to provide the customer's order, it sends a confirmation to the customer and then it delivers the order.
- The manufacturer could not be in possession of all the parts necessary to deliver after contacting the supplier. In this case, it must open an auction with a *bidder* to obtain the missing pieces.

– If the manufacturer has finally obtained all the parts needed, the same actions as before are done: it sends to the customer a confirmation, and then delivers the order.
– If still missing pieces it should reject the order.

Fig. 2 illustrates in BPMN's collaboration diagram the scenario where the public interactions (with no sequencing order) between the actors are presented.

3.2 The Domain Meta-model

Fig. 3 shows the domain meta-model where the fundamental elements of a choreography are presented.

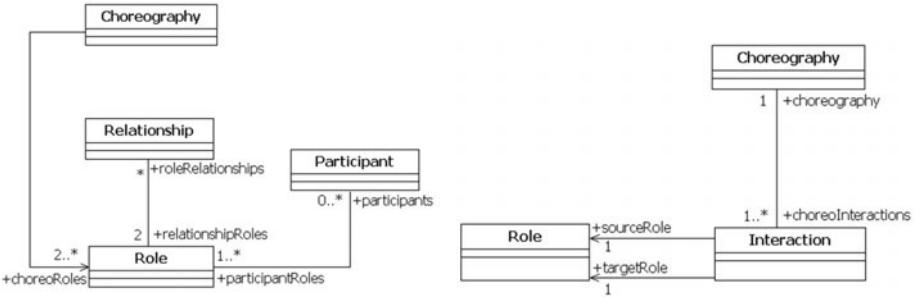


Fig. 3. The domain Meta-Model

From a structural point of view, let us consider the communication protocol (*Choreography*) that has a set of roles (*Role*) linked in two by two relationships (*Relationship*). The relationship represents the existence of a previous knowledge between both roles. A role is an abstract entity (e.g. manufacturer and supplier) played by a participant (*Participant*) that is the concrete entity (e.g. "John" or "ComputerSeller.com"). A participant may play multiple roles in a choreography and a role can be played by several participants (at design time).

From a behavioral point of view, a choreography is defined as a set of interactions (*Interaction)* between two roles (e.g an interaction can be the request of some missing pieces from the manufacturer to the supplier). In an interaction there is a role transmitter (*sourceRole*), in this case the manufacturer, which is the one that starts the interaction and a role receiver (*targetRole*), in this case the supplier.

The modeled scenario illustrating the domain meta-model concepts is presented in Fig. 4. It represents the structural view but we add interactions. Arrows indicate the "sense" of the interactions i.e the source role and the target role within the interaction. This example is only an illustrative representation that helps us to understand the meta-model. We could also consider the possibility of separating both views in different models but due to the simplicity of the

example we decided to illustrate all concepts in the same representation. Graphical notation is an issue of future work and this is not discussed in this article

We retain from this layer that the first elements to be identified in a choreography are the roles and optionally the participants playing that roles. Then, a set the interactions between roles (behavioral) that implies defining their relationships in a static way.

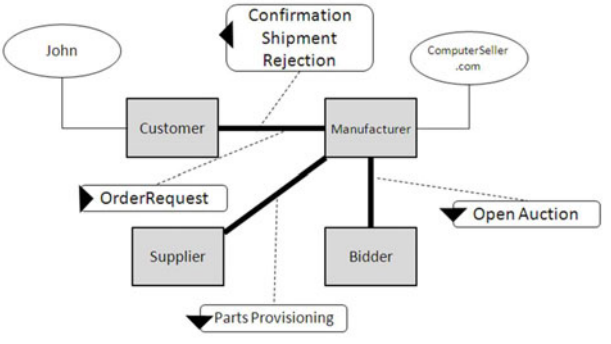


Fig. 4. Structural view representing the scenario + Interactions

3.3 The Analysis Meta-model

The analysis meta-model presented in Fig. 5 is the layer in which are represented the elements that must appear in every choreography implementation. In particular, we introduce the control flow mechanisms and the messages exchanged between the choreography participants via the service operations.

Analysis meta-model is explained through the same example but extended. A participant that plays a role must provide the services (*Service*) defined for that role to respect the choreography. Note that we consider a service as a logical entity i.e the way to access the set of defined operations (*Operation*). Each operation defines a request message (*request*) and optionally, a response message (*response*) and error messages (*errorMessages*).

In the behavioral view, we introduce some new elements. As a choreography is a set of ordered interactions between roles, we need mechanisms to compose and order them. We introduce the activity class (*Activity*) as a generalization of control flow activities (*ControlFlowActivity*), and interactions (*Interaction*). A *ControlFlowActivity* can be choice (*Choice*), parallel (*Parallel*) and sequence (*Sequence*) activities. When an operation is invoked in an interaction, it has to be an operation defined within the *targetRole*'s services. For example, the customer could invoke an operation "manageOrderRequest" provided by the manufacturer in the service "CustomerToManufacturerService". A request message defined in this operation could be for example "requestOrderMessage" and a response message "AcknowledgmentMessage".

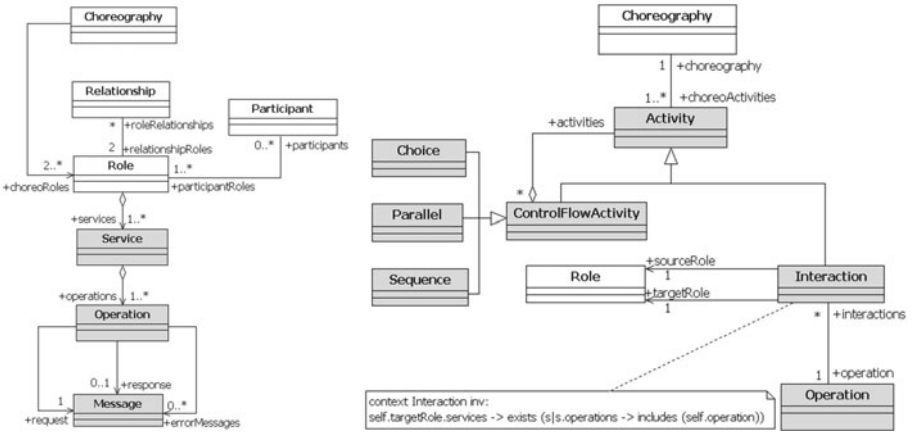


Fig. 5. The Analysis Meta-Model

The domain meta-model and the analysis meta-model respectively defined in Fig. 3 and Fig. 5 are close. The domain meta-model is included in the analysis meta-model (classes in white) where some domain classes are refined to go into details in the analysis meta-model as the *Role* or the *Interaction*.

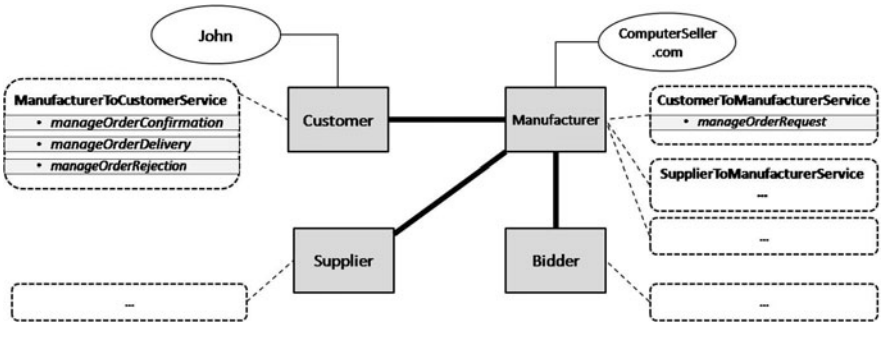


Fig. 6. Structural view of the scenario - Analysis Model

Fig. 6 and Fig. 7 partially model the scenario presented in section 3.1. Fig. 6 represents the structural view where the services provided by each role might be defined. Operations are also defined but messages are not specified as we want to stay simple. Our behavioral model in Fig. 7 is inspired in UML's sequence diagrams. Interactions (arrows) between roles (rectangles), operation calls (text above the arrows) as well as sequencing (life line and numbers) or choice (alt) can be represented. This example illustrates in an easy manner our analysis meta-model concepts.

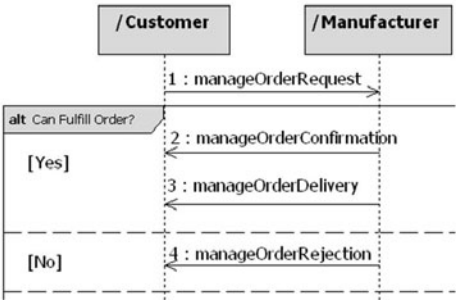


Fig. 7. Behavioral view of a portion of the scenario - Analysis Model

We observe that the analysis meta-model can help users to understand the communication between roles, the interactions sequencing, and the operations required for each role depending on the implemented service.

3.4 The Design Meta-model

Our design meta-model is based on WS-CDL syntax. Fig. 8 shows the defined packages and it's dependencies. These packages are the starting point of our bottom-up approach. In general, the main package classes have a *corresponding meta-class* in the higher abstraction levels as for example *Interaction*, *Role* or *Choreography*. The complete design meta-model is presented in the Appendix.

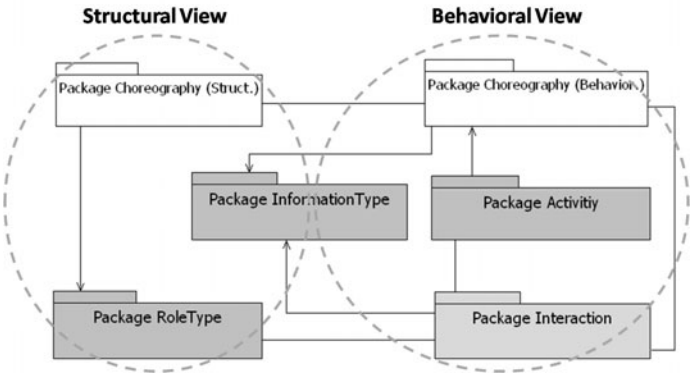


Fig. 8. Design meta-model package overview and dependencies

Code 1.1 and Code 1.2 represent simple WS-CDL's pieces of code that correspond to the scenario. Code 1.1 is what we consider the static definition of WSC-DL's choreographies where roles, relationships, participants and message types are defined. Code 1.2 illustrate the behavioral definition of the choreography where sequencing of interactions and variable declarations are depicted.

This behavioral section is identified by "< *choreography* >" label in WS-CDL syntax. More detailed WS-CDL's examples are found in [17].

These examples show that a WS-CDL skeleton could be generated from the analysis model. Regarding the static section, roleTypes (lines 1-9) , participantTypes (lines 11-16) and relationshipTypes (lines 19-22) could be completed. In the behavioral section, interaction sequencing and role's communication could also be completed almost entirely. However, a very simple example is shown. To fully complete a WS-CDL file, we might need the design meta-model. For example, we might complete the message's type exchanged between roles (lines 25-26) and the channelTypes (lines 28-36) to define the way of accessing roles. In the behavioral section, we might also define variable's declaration and treatment (lines 2-7) or the XPath queries to reference variables within the XML file (lines 16-17).

```
1 <roleType name="Costumer">
2     <behavior name="CostumerService" interface="ManufacturerToCostumerService">
3     </behavior>
4 </roleType>
5 <roleType name="Manufacturer">
6    <behavior name="ManufacturerService" interface="CostumerToManufacturerService">
7    </behavior>
8     ...
9 </roleType>
10
11 <participantType name="John">
12     <roleType typeRef="Costumer"/>
13 </participantType>
14 <participantType name="ComputerSeller.com">
15     <roleType typeRef="Manufacturer"/>
16 </participantType>
17 ...
18
19 <relationshipType name="CostumerToManufacturerRel">
20     <roleType typeRef="Costumer"/>
21     <roleType typeRef="Manufacturer"/>
22 </relationshipType>
23 ...
24
25 <informationType name="OrderRequestType" type="OrderRequestMsg">
26 </informationType>
27
28 <channelType name="CostumerToManufacturerChannel">
29     <roleType typeRef="Manufacturer"/>
30     <reference>
31         <token name="tns:URI"/>
32     </reference>
33     <identity type="primary">
34         <token name="tns:id"/>
35     </identity>
36 </channelType>
```

Code 1.1. Example of WS-CDL's code (structural)

```
1 <choreography name="computerPurchaseChoreography">
2 <variableDefinitions>
3    <variable name="orderRequest" informationType="tns:OrderRequestType"
4              roleTypes="tns:BuyerRole_tns:SellerRole">
5    </variable>
6  ...
7 </variableDefinitions>
8
9 <sequence>
10   <interaction name="OrderRequest" operation="manageOrderRequest"
11                channelVariable="tns:Buyer2SellerC">
12   <participate relationshipType="CostumerToManufacturerRel"
13                fromRoleTypeRef="Costumer" toRoleTypeRef="Manufacturer"/>
14
15      <exchange name="OrderRequest" informationType="OrderRequestType">
16         <send variable="cdl:getVariable()"/>
17         <receive variable="cdl:getVariable()"/>
18      </exchange>
19
20      <exchange name="ErrorExchange" informationType="tns:ErrorConfirmationType">
```

```
21          <send variable="cdl:getVariable()"/>
22          <receive variable="cdl:getVariable()"/>
23        </exchange>
24     </interaction>
25     <choice>
26        <sequence>
27           <interaction name="Rejection" operation="manageRejection"
28                        channelVariable="">
29        ...
30           </interaction>
31        </sequence>
32        <sequence>
33           <interaction name="Confirmation" operation="manageConfirmation"
34                        channelVariable="">
35              ...
36           </interaction>
37           <interaction name="Shipment" operation="manageShipment"
38                        channelVariable="">
39              ...
40           </interaction>
41        </sequence>
42     </choice>
43 </sequence>
44 </choreography>
```

Code 1.2. Example of WS-CDL's code (behavioral)

We can imagine the huge quantity of code that should be managed when defining complex choreographies by regarding this simple (and not completed) pieces of code. Abstraction is needed to manage choreography complexity in a progressive way. To achieve the goal of approaching business world and service composition world, abstraction layers and different viewpoints should be a main concern to make this concept understandable and exploitable for all the actors participating in an inter-organizational business process set up.

4 Conclusion and Discussion

In this paper we have presented a service choreography meta-model based on modeling techniques as abstraction layers and views separation. Fig. 9 shows a global overview of our approach and future work. A bottom-up approach helps to understand and validate a choreography. A top-down approach helps to implement the final executable processes for each party avoiding ambiguities. We locate the domain and analysis layers into business processes and design layer into the service composition world. We represent the separation as a wavy area as the separation is fuzzy.

By transformation rules, we could move from one layer to another. Therefore, transitions between the abstraction layers manage in a gradually way the gap between business processes and service composition implementations. Formalization of this transformations are envisaged for future works so the passage between layers could be done the more automatically as possible.

We want to extend our approach creating a graphical notation corresponding to the meta-model. As in the Pi4SOA tool [13], we think that the two-views separation is an important aspect to manage the complexity of a service composition model. More generally, the formalization of the three abstraction layers and the views split will help us in respecting the *Principle of Complexity Management*, the *Principle of Cognitive Integration* and the *Principle of Cognitive Fit* defined by Moody for the physics of graphical notation [9]. This part of our work could

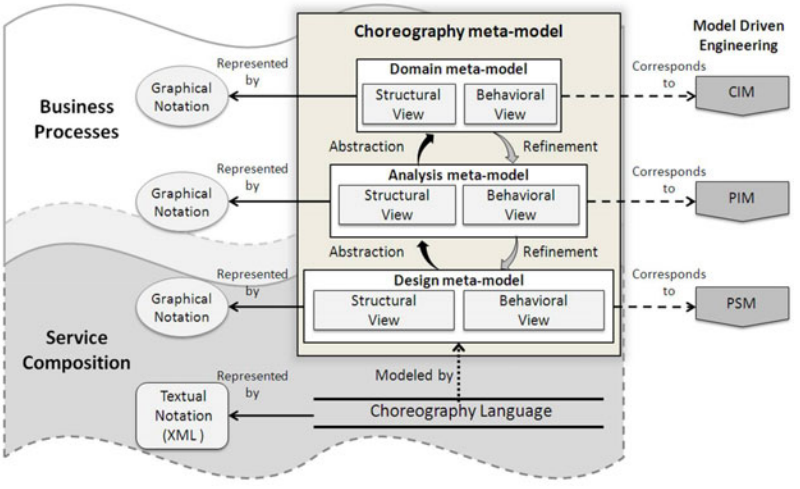


Fig. 9. MDE correspondences and future graphical notation work

be clearly compared to the OMG's BPMN 2 [12] work, which incorporates a new graphical notation for choreographies. It means that choreography could be an important concept for business process community.

We also observe in Fig. 9 an obvious correspondence with the paradigm of Model Driven Engineering (MDE) [15]: our domain meta-model layer is equivalent to Computation Independent Model (CIM), the analysis meta-model layer correspond to a Platform Independent Model (PIM), and the design meta-model layer based on WS-CDL is an example of Platform Specific Model (PSM) in MDE. Between the different layers, refinement and abstraction relationships will be implemented by MDE transformations. So this meta-model approach is our first step to help designers and developers in bridging the gap between business process and service composition implementations.

Acknowledgments

We would like to thank Gabriel Pedraza, German Vega, Agnes Front and Aurelien Faravelon for their helpful comments. We also thank the international scholarship program FARO Global for their support.

References

1. Van der Aalst, W., Benatallah, B., Casati, F., Curbera, F., Verbeek, E.: Business process management: Where business processes and web services meet. Data and Knowledge Engineering 61(1), 1–5 (2007)
2. Barker, A., Walton, C., Robertson, D.: Choreographing Web Services. IEEE Transactions on Services Computing 2(2), 152–166 (2009)

3. Barros, A., Dumas, M., Oaks, P.: A critical overview of the web services choreography description language. BPTrends Newsletter 3 (2005)
4. Barros, A., Dumas, M., Ter Hofstede, A.: Service interaction patterns: Towards a reference framework for service-based business process interconnection. Faculty of IT, Queensland University of Technology FIT-TR-2005-02 (2005)
5. Decker, G., Barros, A.: Interaction modeling using BPMN. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 208–219. Springer, Heidelberg (2007)
6. Decker, G., Kopp, O., Barros, A.: An introduction to service choreographies. Information Technology 50(2), 122–127 (2008)
7. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: IEEE International Conference on Web Services, ICWS 2007, pp. 296–303. IEEE, Los Alamitos (2007)
8. Decker, G., Kopp, O., Leymann, F., Weske, M.: Interacting services: from specification to execution. Data & Knowledge Engineering 68(10), 946–972 (2009)
9. Moody, D.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering, 756–779 (2009)
10. OASIS: Web services business process execution language v2.0. (2007), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
11. OMG: Business process management notation (v1.2) (2009), http://www.omg.org/spec/BPMN/1.2/
12. OMG: Business process management notation (beta 2) (2010), http://www.omg.org/cgi-bin/doc?dtc/10-06-04
13. Ross-Talbot, S., Brown, G., Honda, K., Yoshida, N., Carbone, M.: Pi4soa technologies fundation, http://sourceforge.net/apps/trac/pi4soa/wiki
14. Ross-Talbot, S., Brown, G., Honda, K., Yoshida, N., Carbone, M.: Soa best practices: Building an soa using process governance (2009)
15. Soley, R., et al.: Model driven architecture. OMG white paper 308, 308 (2000)
16. Srivastava, B., Koehler, J.: Web service composition-current solutions and open problems. In: ICAPS 2003 Workshop on Planning for Web Services, vol. 35. Citeseer (2003)
17. W3C: Web services choreography description language version 1.0 - w3c candidate recommendation (2005), http://www.w3.org/TR/ws-cdl-10/
18. Weidlich, M., Barros, A., Mendling, J., Weske, M.: Vertical alignment of process models – how can we get there? In: Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) Enterprise, Business-Process and Information Systems Modeling. LNBIP, vol. 29, pp. 71–84. Springer, Heidelberg (2009)
19. WSC: Web services choreography working group (2002), http://www.w3.org/2002/ws/chor/
20. Zaha, J., Barros, A., Dumas, M., ter Hofstede, A.: Let's dance: A language for service behavior modeling. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 145–162. Springer, Heidelberg (2006)

A Appendix

The UML class diagrams in Figs. 10 and 11 provide a high-level overview of WS-CDLs meta-model. Fig 10 describes the structural view while 11 describe the behavioral view. These meta-models are based upon the WS-CDL meta-model in [3]. They represent our today's design meta-model.

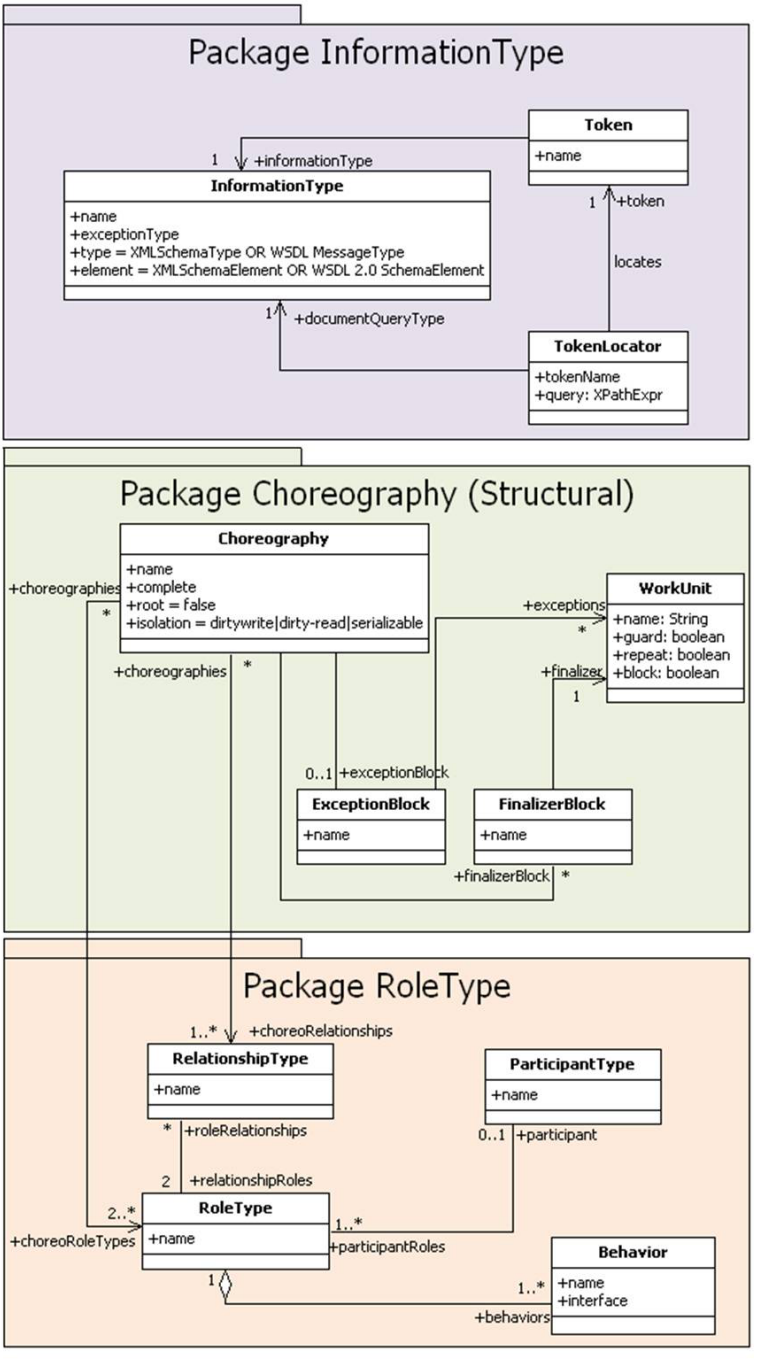


Fig. 10. WS-CDL design meta-model - structural view

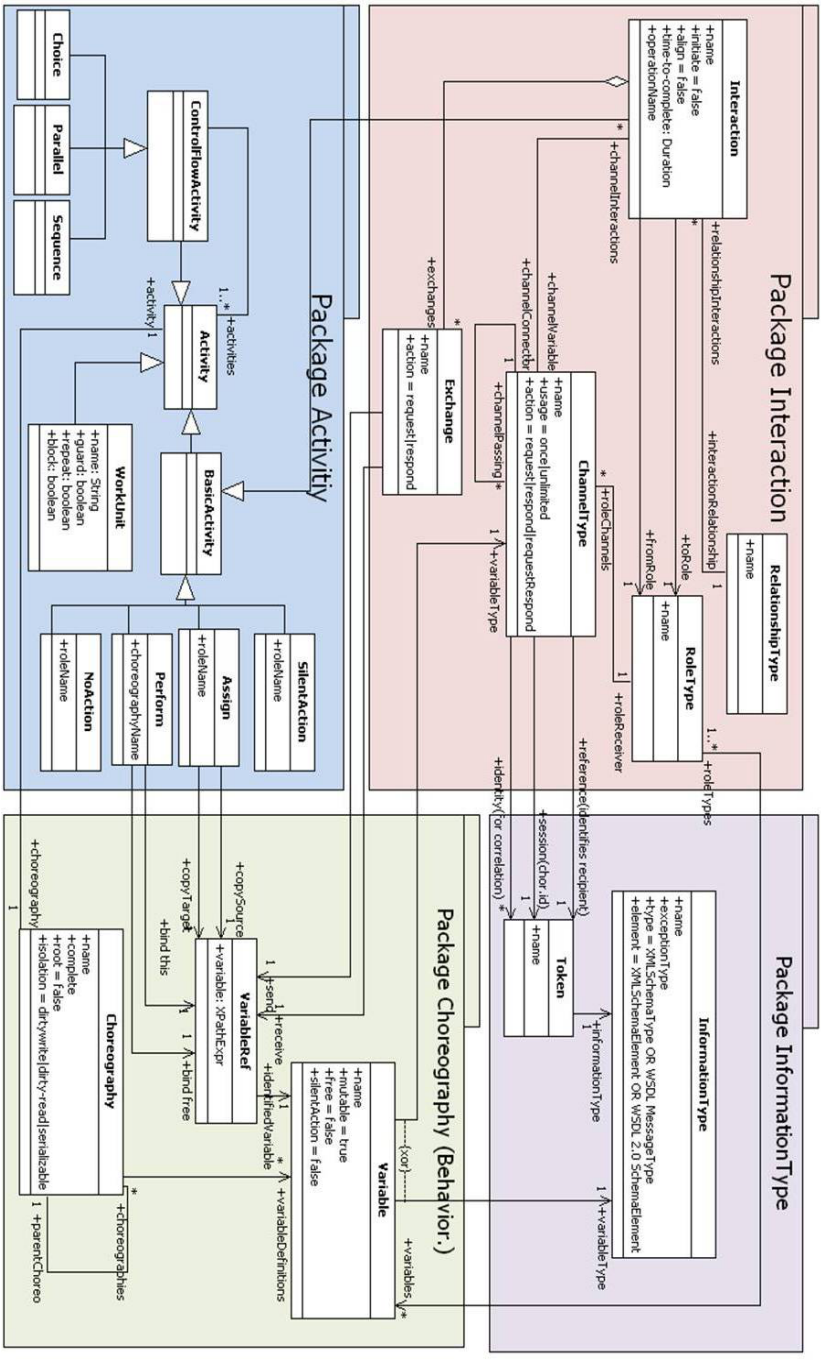


Fig. 11. WS-CDL design meta-model - behavioral view

Towards Construction of Situational Methods for Service Identification

René Börner

ProcessLab, Frankfurt School of Finance & Management,
Sonnemannstraße 9-11, 60314 Frankfurt am Main, Germany
r.boerner@fs.de

Abstract. The service-oriented paradigm plays an increasingly significant role in designing and governing IT architectures in organizations. The identification of services belongs to the most important parts of the service management life-cycle and is essential for the successful implementation of service-oriented architectures (SOA). However, existing methods for service identification mostly ignore situation-specific factors for such projects. Situational method engineering can be used to design a meta method to support the development of situation-specific methods for service identification. Based on a literature review and two case studies, this paper elaborates on context factors and SOA implementation goals being constituting elements of situations. These situations are tailored to the service identification domain. Applying this meta method in concrete project situations will help to engineer appropriate methods for service identification.

Keywords: Situational method engineering, service-oriented architectures, service identification, context factors, meta method, SOA implementation goals.

1 Introduction

Service orientation is currently a dominating paradigm for enterprise and IT architectures. Service-oriented architectures promise a greater flexibility of IT and a faster adoption to changing business needs. The identification of services is one of the most important steps for successful SOA implementations and many authors have developed methods for this purpose (for an overview see [1]). Interestingly, most of these methods are based on a one-fits-all approach and do not consider a configuration of methods depending on different circumstances.

The field of situational method engineering (SME) offers opportunities to overcome these shortcomings. A central aspect behind it is that a fixed method is not suitable for all situations that occur in reality. Thus, methods have to be adaptable to different kinds of situations. This paper elaborates on context factors and SOA implementation goals to identify situations in the field of service identification. These are necessary to support a reasonable configuration of fragments that are developed as part of the meta method for the configuration of methods for service identification. Identifying situations is pivotal for this meta method and the focus of this paper.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 204–218, 2011.

Designing suitable fragments complements the meta method and will be discussed briefly herein.

The paper is structured as follows: Section 2 discusses related work in the fields of SME and service identification, outlines the research design and describes the scope of this paper. In section 3, context factors and SOA implementation goals – both being defining parts of a situation – are elaborated in detail. The design of method fragments for service identification is presented in section 4. Section 5 reflects on limitations, proposes avenues for future research and provides a conclusion.

2 Foundations of a New Meta Method

This section discusses related literature in the field of SME and service identification. The research process used to derive the context factors is outlined. Furthermore, scope and goals of the meta method are defined.

2.1 Situational Method Engineering and Configurability in Existing Approaches

It is commonly accepted that no universal method constructed at time (t_1) can fit every conceivable situation in which it is applied in time (t_2) [2]. Actually, it is quite improbable that a rigid method developed from theory is applicable in a concrete setting without modification [3] and therefore, the concept of SME emerged (see e.g. [4]). The central aspect behind it is that a fixed method is not suitable for all situations that occur in reality. Thus, methods have to be adaptable to different kinds of situations. To support this adaptability, smaller parts of a method – so called method fragments – are created and can be composed depending on the situation at hand [5]. Method fragments consist of the four elements activity, technique, role, and result [6].

Unfortunately, the term method fragment is inconsistently used in literature [7]. Ågerfalk et al. [8] define method fragments as "standardized building blocks based on a coherent part of a method" (p. 360). A situational method can be constructed by combining a number of method fragments. For the purpose of this paper, any reasonable combination of method elements representing a coherent part of a method shall be referred to as method fragment [8].

According to Bucher et al. [9], there are two adaptation mechanisms to engineer a situational method, namely *situational method configuration* and *situational method composition*. Situational method configuration follows the so called adaptive principle. This means that a base method is created at design time (t_1) and configured in certain contexts at time (t_2). For situational method configuration, situational changes to a base method have to be foreseen and planned when a situational method is developed at time (t_1). In contrast, situational method composition provides for a spontaneous combination of method fragments (orchestration) that does not have to be foreseen at (t_1). There is no pre-defined base method that is adapted. Instead, method fragments are combined and aggregated as required at (t_2). Börner [10] suggests a third possibility, in which pre-composed methods are assigned to situations in (t_2). Although methods are defined in (t_1) already, there is no pre-configured base method in this approach.

Many authors agree that characteristics of a project have to be defined in order to describe a situation [11-13]. Still, according to [9] they do not explicitly say what constitutes a situation. For the purpose of this paper, Börner's [10] concept of a situation is used. It defines *context factors* and *SOA implementation goals* as two determining factors that constitute a situation.

The analysis and comparison of existing service identification literature is the basis for the development of a meta method for situational methods conducted in this paper. Additionally to the literature analyzed by [1], two currently published approaches [14, 15] were included in this paper. All approaches were examined considering their description of activities, roles, techniques and results as well as the configurability of the presented methods. The first four criteria were chosen because they are commonly used elements for methods and method fragments [6, 16]. Configurability is frequently regarded important in SME literature. However, in the field of SOA and especially in service identification a lack of configurability of methods can be stated.

Since a*ctivities* are the focus of all compared approaches, they are described precisely in most cases. Many *techniques* and intermediate *results* are usually provided as well. The most striking feature is the almost complete absence of *roles*. Although many authors discuss the importance of business and IT alignment, all compared service identification approaches tend to underestimate the significance of properly assigning roles to respective activities and techniques. Roles will play an important role when method fragments are assigned to situations.

The *configurability* describes the possibility to choose adequate fragments that suit the situation at hand and arrange them in an adequate sequence. The latter means that activities used in an approach do not have to be executed in a linear order but can be used iteratively and hence allow for loops or iterations. Although hints on possible configurations can be found in some places, none of the authors explicitly incorporates the former into their approach. Since the goal of this paper is to guide the engineering of situational methods, configurability of service identification methods will be considered a crucial feature.

2.2 Research Design

The development of a meta method for service identification methods presented herein is based on a hybrid research approach combining several research methods to gain a richer understanding of the topic [17]. Construction of the meta method supported by SME belongs to the realm of *design science*. The derivation of relevant context factors builds on both *desk research*, i.e. literature reviews, and *case study research*. Since the identification of relevant context factors is an explorative goal, a case study approach was deliberately chosen to give the results an empirical grounding. Case studies are appropriate in this respect because they "provide descriptions of phenomena" [18]. Furthermore, case studies are particularly relevant for research in its "early, formative stages" [19] which applies to the field of SOA [20].

The case studies were conducted in two SOA implementation projects in Australian companies providing completely different environments. One of the companies is a small data provider; the other is one of Australia's biggest insurance companies. The significantly diverse settings of both cases opened up a continuum [21] of instantiations for identified context factors, i.e. their parameter values. At the insurer,

researchers conducted an action research study. They actively participated in the project and helped test and apply a service analysis & design methodology developed by them previously. In the second case study, the most important sources of evidence have been interviews that were conducted shortly after the project had been completed. The data provider's employees and researchers were the interview partners. The researchers had helped the data provider to implement an SOA in order to enable the retrieval and analysis of heterogeneous data from different sources (grid environment) spontaneously in an unforeseeable fashion (ad-hoc).

The interviews have been transcribed afterwards and analyzed along with all other documentation and reports. In an iterative approach, relevant context factors were determined based on this data and compared with related literature. The identification of such factors (concepts) was conducted by employing techniques from grounded theory, for example, open and axial coding [22], and interpretative techniques [23]. Even though these coding techniques were not used to their fullest extent, the general approach and respective tools supported the assignment of statements from the interviews and documents to concepts. The goal was to detect relevant particulars within the case data and to identify relevant concepts [24], i.e. context factors.

2.3 Goals and Scope of the Meta Method

Due to the lack of configurability in existing approaches, this paper argues that situational method engineering can support methods for service identification that suit certain project situations and are thus situation-specific. Particularly, context factors (including their respective parameter values) and SOA implementation goals that jointly determine a situation are the focus. They provide the basis for the intended meta method that is subject to ongoing research.

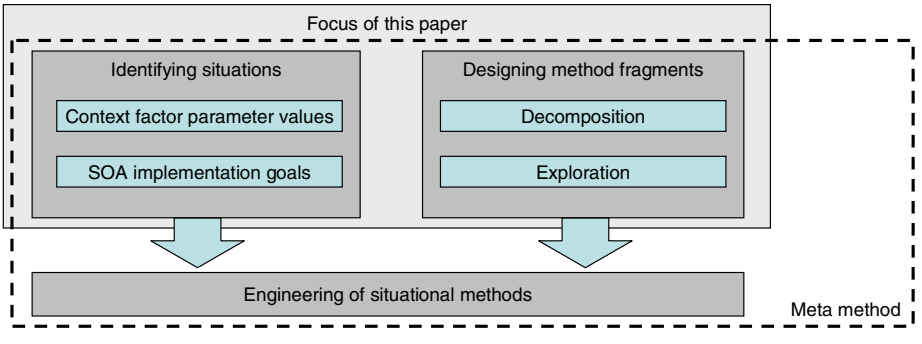


Fig. 1. Scope of the Meta Method and Focus of This Paper

This meta method encompasses the identification and description of context factors, their value parameters and SOA implementation goals. Together, combinations of these parts constitute a situation. Every instantiation of a method will rely on the situations that will be developed in section 3 since context factors, value parameters and SOA implementation goals belong to the service identification domain. Thus, the latter are design elements of this meta method. Apart from situations, the meta

method also includes descriptions of method fragments. Principles of fragment design will be outlined and shown exemplarily in section 4.

Figure 1 illustrates that the meta method encompasses the identification of situations and the design of method fragments. Both are necessary for the configuration of situational methods. Focus of this paper is the identification of situations in the domain of service identification based on relevant context factors and SOA implementation goals. Moreover, the design of method fragments as part of a meta method will be shown exemplarily herein. Therefore, two ways of method re-engineering, namely decomposition and exploration, are presented. A generally valid description of how to engineer situational methods in any conceivable situation is left to further research.

3 Identifying Situations

Following [10], the combination of context parameters and SOA implementation goals determines situations. Figure 2 illustrates the five necessary steps (a) to (e) to identify situations. These steps are introduced briefly herein (for a more detailed explanation see [10]):

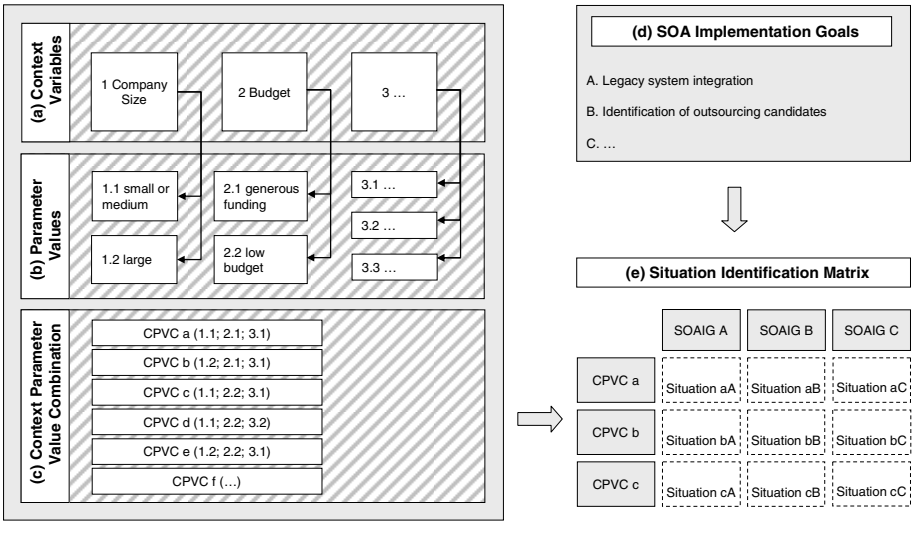


Fig. 2. Five Steps to Identify Situations (following [10], p. 5)

(a) The context variables (contingency factors) that can influence the SOA implementation project at hand are defined.
(b) The context variables identified in (a) have certain parameter values (instantiations) that are defined in this second step.
(c) All context parameter values are combined with one another. These context parameter value combinations (CPVC) serve as one input for the situation identification matrix.

(d) Possible SOA implementation goals are identified in this step. They are the second input for the situation identification matrix.
(e) The situation identification matrix illustrates all possible combinations of CPVCs and SOAIGs retrieved in steps (c) and (d).

Context factors, respective parameter values and SOA implementation goals that are relevant to the service identification domain, and therefore an important part of the meta method presented herein, will be introduced in the following subsections. A combination of concrete parameter values (e.g. a small company with low budget, no industry-specific legal restrictions producing only one product) and an SOA implementation goal (provision of services for third parties) constitutes a specific situation.

3.1 Context Factors

Based on related literature and two recently conducted case studies described in section 2.2, the importance of the most influential context factors is discussed in the following. Although these context factors might not be unique to the field of SOA, their consideration in service identification methods will improve applicability of the latter.

It is a generally held belief that the ***company size*** is a considerable contextual factor in many kinds of software projects [25] such as service identification in SOA implementation projects. Whereas Sedera [26] proposes three classes of company size, this paper follows Welsh and White [27] who suggest only two. Hence, companies are differentiated into *small and medium-sized enterprises* on the one hand and *large companies* such as multi-national enterprises on the other hand. The former shall comprise organizations with up to 250 employees [28].

The ***budget*** of a project plays an important role when it comes to choosing necessary steps and general proceeding of a method for service identification. Generally, a generous budget that allows for an extensive time frame provides the opportunity for a thorough and systematic application of identification methods. One would expect the utilization of many techniques in order to ensure a high quality of implemented services. A detailed analysis of available strategic and technical documents would be typical in such circumstances. Literature confirms that the budget has implications on the number of available staff, the time pressure and the possibility to incorporate external help from consultants [29].

Some activities have to be carried out regardless of the budget. Still, there might be obligatory as well as optional techniques which support the activity. Consequently, an exhaustive use of techniques would only be chosen if financial resources are easily available. The parameter values *generous funding* and *low budget* will thus be used for this context variable.

Depending on a company's strategy, services can be provided for different ***service consumers***, for example other divisions (internally), third parties (externally) or both. Strategically important services that lead to a competitive advantage in the market should only be available to end-consumers but not to competitors. A clear distinction between services offered to internal customers, other companies in a value chain or end-consumers is indispensable. If it is known in advance that a service will be offered internally only, a number of activities and results such as the creation of an inter-organizational service map are not applicable in this situation. Moreover, there

might be legal constraints that only apply if services are offered to third parties and thus demand an examination of these regulations. An analysis of consumer interaction is always essential whereas the "line of visibility" is much more important if services are exposed to external customers [30].

For the purpose of this paper, service consumers will be divided into *internal* and *external consumers*. Additionally, the case of a service being offered to *both* is considered.

Skills and experience with both service-oriented architectures and business process management significantly influence the proceeding of service identification. Limited SOA experience on the side of employees often leads to a technical SOA understanding. Project teams that are more familiar with the service-oriented paradigm are more likely to succeed in combining technical aspects with a Business Process Management perspective in mind. Hence, software services that support business processes or at least sub-processes can be the goal of their analysis. Their identification usually includes activities related to the analysis of process models and involves not only IT-related, but also business-related staff roles. If the service identification is limited to a rather technical point of view, the set of method fragments to be considered will therefore be a different one.

The configuration of situational methods and choice of fragments is affected, if for example certain roles cannot be occupied by available employees. Limited employee skills can necessitate external support by consultants. Although this enables the application of certain fragments, this option might be limited by the project budget and might therefore not be feasible. Parameter values for this context factor are *SOA skills available, BPM skills available, both skills available* and *none available*.

Furthermore, the ***SOA maturity level*** a company has achieved is seen as a further influential factor on the delivery strategy of SOA [31]. Thus, it plays an important role in the configuration of methods for service identification.

SOA maturity models are used to classify the status of SOA implementations within a company. This paper will use the Service Integration Maturity Model (SIMM) [32] to distinguish advanced organizations with *level 4 to 7* from less mature organizations (*level 1 to 3*). The former are likely to use more sophisticated and strategy-oriented fragments. The latter usually use more technically-oriented techniques and thus other fragments.

On the one hand, ***compliance*** issues can arise from legal obligations and regulatory restrictions. These differ among countries and especially companies that operate in more than one country have to consider legal demands arising from that. In many countries, all companies have to obey certain rules as far as the confidentiality of customer data is concerned. Additionally, some industries such as banking or pharmaceuticals have to adhere to special regulations. Finally, regulations can arise from the fact that a company is listed on a stock exchange, i.e. it also depends on its legal form. On the other hand, internal policies may require corresponding method fragments that address issues like service ownership. Three parameter values will be used for this context factor, namely *standard legal compliance, special regulations due to industry, legal form or international operations* and *internal policies*.

Another important context variable is the *existence* of a designated ***IT department*** and thus the degree of centralization of the IT infrastructure. In a small company that lacks an IT department, methods have to be adapted to accommodate for this

circumstance. Larger organizations usually have such an IT division or are structured along the lines of business. On the one hand a high degree of centralization or the existence of a central division supervising and governing IT implementation throughout a company usually leads to more transparency. Frequently, at least some information on applications and data is readily available. This can be used as input for service identification method fragments. On the other hand, some fragments demand certain roles such IT administrators or newly composed units consisting of business and IT employees (see also [33]). In a small company that lacks an IT department, these method fragments are frequently not applicable.

Table 1. Context Variables and Respective Parameter Values

Context Variable	Parameter Value
1 Company size	1.1 Small or medium-sized enterprise
	1.2 Large company
2 Service consumers	2.1 Internal consumer
	2.2 External consumer
	2.3 Internal and external consumers
3 Budget	3.1 Generous funding
	3.2 Low budget
4 Skills and experience	4.1 SOA skills available
	4.2 BPM skills available
	4.3 Both skills available
	4.4 None available
5 SOA maturity level	5.1 SIMM level 1-3
	5.2 SIMM level 4-7
6 Compliance	6.1 Standard legal compliance
	6.2 Special regulations
	6.3 Internal policies
7 IT department	7.1 Existent
	7.2 Not existent
8 Interaction	8.1 Customer interaction
	8.2 Employee interaction
	8.3 Customer and employee interaction
9 Organizational structure	9.1 One product company
	9.2 Multiple product company

Varying degrees and forms of ***interaction*** with both customers and employees necessitate the use of different method fragments. In some cases employees are not directly involved in service delivery because the services are very fine-grained and fully automated. The coarser-grained services are, the greater is the possibility that they are only semi-automated or manual and subsequently interact with employees. Customer interaction can be of high importance when the composition of services by the end user is a primary goal. In general, a customer interaction can be obligatory in some places or can happen "on demand" if required [34]. If customer interaction is a major issue for the identification of services in a situation at hand, respective method fragments (e.g. swim lane diagrams that show interfaces to customers) are crucial for a successful implementation. Thus, *customer interaction*, *employee interaction* and a *combination of both* are differentiated for the purpose of this paper.

In a company specialized on one product only, an analysis of a service's reusability is trivial in most cases. The same analysis is much more complex when looking at companies with a wide range of products. An organization can, e.g., be structured by products (business lines), regions, functions or customer groups [35, 36]. Even a multidimensional structure combining two or more dimensions of the above is not uncommon. Thus, the ***organizational structure*** can be an important factor when it comes to service identification. Herein, *one product companies* and *multi product companies* are differentiated.

Table 1 gives an overview of the context variables used and their respective parameter values. After identifying the context factors that are one part of a situation, the next section will elaborate possible goals for the implementation of service-oriented architectures that are the second constituting part of a situation.

3.2 SOA Implementation Goals

The second constituting element of a situation are SOA implementation goals. Depending on the purpose of an SOA implementation, the identification of services can necessitate the application of different method fragments. Many such goals can be found in related literature and the case studies also confirmed some of them. In the following, these goals and their influence on a situational method configuration will be discussed.

The ***integration of legacy systems*** is a frequently mentioned goal of SOA implementations [37, 38]. Especially in medium-sized and large enterprises, IT architectures have developed over years or even decades. In the absence of a central governing body, manifold isolated applications were developed and implemented which led to a plethora of problems. New functionalities and updates have to be made separately for each system which causes high maintenance costs. In some cases, it is difficult to find specialists who are able to administer for instance cobol code. Due to their restricted function-oriented view, employees do not know about IT systems of other divisions. This redundancy causes high costs because of unnecessary licensing fees.

Hence, integrating existing applications plays a major role in enterprise IT architectures and is one of the reasons for SOA implementation projects. In this case, techniques such as asset analysis and results that illustrate dependencies of the existing IT infrastructure are crucial parts of the service identification. The knowledge of IT experts about technical interfaces is indispensable.

The ***identification of outsourcing candidates*** is another goal for SOA implementations [39]. In this case, costs, performance and strategic relevance of services must be analyzed. On the one hand, based on a business process analyses the exact scope of the outsourcing activity has to be defined. A strategic make-or-buy decision determines which parts of the process are performed within the organization and which parts shall be outsourced to service providers. On the other hand, an outsourcing candidate needs clearly defined technical interfaces. Inputs and outputs of automated services provided by a third party have to be explicated in service level agreements. These outsourcing considerations demand fragments that produce for instance inter-organizational service maps and incorporate strategic aspects as well as detailed technical descriptions.

The ***agility and flexibility of business processes*** is a competitive advantage and strongly tied to the concept of SOA [40]. An alignment of business and IT is a necessary precondition to achieve this flexibility. Therefore, a company's strategy, i.e. a business process perspective, has to be considered. An enterprise-wide governance of the IT infrastructure is indispensable to provide for this agility. Hence, fragment results such as service ownership models [41] have to be used.

In contrast to an enhanced flexibility on process services level, the ***standardization*** of basic services is meant to avoid redundancies in development and maintenance of IT and thus to reduce costs significantly [42]. The goal is to improve efficiency by reusing a service in as many processes as possible. However, a customer should not be limited in his choice of varieties. A faster processing through increased efficiency should lead to a higher customer satisfaction. Therefore, services that directly interact with customers should not be standardized. This makes fragments for the analysis of the line of interaction and the line of visibility indispensable.

A completely different perspective is taken by companies that aim at the ***provision of services for third parties***. The former specialize on a small part of a value chain concentrating on their core competencies. These companies are able to generate economies of scale by providing services for many other companies typically – not necessarily – belonging to the same industry sector. Hence, the focus here is again on inter-organizational and strategic instruments. Services must be easily exposable to third parties, i.e. interfaces have to be well-defined and performance has to be readily measurable. Method fragments should thus concentrate on interaction, interface analysis and the strategic value of providing a service to third parties.

4 Designing Method Fragments for Service Identification

In order to design situation-specific methods for service identification, method fragments that support this identification have to be provided. There are basically two possibilities to design these method fragments [43]. On the one hand, fragments can be re-engineered from existing methods. On the other hand, they can be designed from scratch in case no experience exists, i.e. no fragments or elements can be retrieved from existing approaches.

As shown in section 2.1, literature provides a number of methods for service identification. Although they include many effective method elements and fragments, the lack of configurability is a major shortcoming. Thus, the following design of method fragments will concentrate on re-engineering of existing methods rather than on ad-hoc construction. Ralyté [44] identifies two ways to design method fragments from existing methods, namely *decomposing models from existing methods* and *exploring different possibilities to apply a model* (p. 5). In the following, both will be introduced. Two examples of method fragments will show the applicability to the service identification domain.

4.1 Decomposition

Identifying fragments through decomposition is supposed to be easier than creating new ones through exploration and should thus be the first step. Most of the fragments

that can be found in existing service identification approaches concentrate strongly on the result of activities and are thus strongly product-driven. In these cases the process part – including roles and techniques – has to be conceptualized in order to obtain fragments. In cases where a fragment is identified by process model decomposition, the product part has to be elaborated since the processes are already available [44]. The following is one example of a fragment that was decomposed from an existing method [30] and enhanced as far as the process part is concerned.

Fragment 1: Overview of Existing Process Models

Description:	If there are documented business processes, these should be used for the further analysis to save time and money if possible.
Input:	Meaningful documents of existing business processes from formerly conducted Business Process Management (BPM) projects
Preconditions:	If no BPM projects had been conducted before, it is necessary to identify business processes before using this fragment.
Taken from:	Klose, Knackstedt, Beverungen (2007)
Design:	In the first phase "preparation" of their approach, Klose et al. [31] include the task "prepare existing process models" into their procedure model (p.1804). Since the authors describe activity, techniques and results, the fragment is identified by product decomposition. Only the role to perform the activity has to be added in order to complete the fragment.
Activity:	Preparation
Role:	Employee of the business department
Technique:	Prepare existing process models
Result:	Consolidated and complete set of hierarchical process models, modeling conventions

Fig. 3. Method Fragment 1

4.2 Exploration

After identifying as many fragments as possible in this first step, exploration is used to find additional fragments on the basis of the elements used in existing approaches. Thus, concrete activities, roles, techniques and results found in different sources are extracted and subsequently used to design new fragments. A comprehensive overview of elements cannot be provided herein, but examples for these constituting elements of methods will be given in the following. The most important sources are the literature on service identification and the two case studies described in section 2.

Activities: Many approaches use activities such as *service analysis* and *service categorization*. *Preparation* is also a common activity to be found in literature. However, activities like *goal definition* or *develop SOA strategy* can be found in only one approach, respectively. Some authors use the word activity for very detailed descriptions of how something has to be done. In the definition used herein this would rather be a technique. Furthermore, one and the same activity might have different names in different approaches. This makes consolidation a difficult task.

Roles: Despite their importance, roles only occur in four of the seven compared approaches. Sometimes they are hard to identify as such because the notion of *consumer view* might be used where *employee of the business department* would be a better description. Besides the *employee of the IT department*, roles like *project manager* were important in the case studies. Related literature additionally suggest new roles such as a *service design unit* for certain activities [33].

Techniques: Consolidating all techniques utilized in literature is difficult since most approaches offer plenty of techniques with sometimes overlapping components and scopes. However, there seem to be some typical and wide-spread techniques that are common to many approaches such as *decomposition of business processes* and *asset analysis*. A couple of other techniques that were frequently encountered are *goal service modeling* and *use case modeling*. It is noteworthy that the scope of the listed techniques can differ considerably. Using a *governance questionnaire* is a straight forward and unambiguous procedure with limited scope and little room for interpretation. The *decomposition of business processes* is much more complex and likely to yield different outcomes depending on who actually conducts the task.

Results: Similar to the significant number of techniques, there are many results presented as part of the methods. These results are outputs of respective activities and techniques and can be an input for the next activity. Thus, they are a crucial link between method fragments. The results themselves are quite different in nature and reach from technical *interface descriptions* to comprehensive *SOA strategy documents* or *network models* on an inter-organizational level. *Use cases, reference processes* and *activity diagrams* are examples for other results.

Fragment 2: IT Governance Analysis

Description:	An organization can have manifold demands when it comes to implementing new IT infrastructures. Using agile methods for example could be one imperative. Technical restrictions, programming language, interfaces or naming conventions can impose restrictions on IT projects. Hence, these IT governance issues can be covered by a questionnaire and are incorporated in this fragment. Employees of the business and the IT department jointly forming a so-called service design unit (SDU) form an important role to successfully design services based on this questionnaire.
Input:	Information on IT governance, IT strategy, SOA strategy
Preconditions:	Derived from an organization's strategy, the IT strategy must be defined and documented before an analysis for service identification can be performed. Often, conventions and principles regarding the implementation and development of IT are not explicated in readily available documents. Therefore, it might be necessary to interview (IT) managers in order to retrieve necessary information.
Taken from:	Klose, Knackstedt and Beverungen (2007), Kohlborn, Korthaus, Chan and Rosemann (2009), Arsanjani, Ghosh, Allam, Abdollah, Ganapathy and Holley (2008), Kohlmann and Alt (2007), Alter, Börner and Goeken (2009)
Design:	Since the scope of this fragment quite wide, elements have been selected from different approaches. Domain decomposition for instance could be found in three existing methods. Special roles and techniques such as an SDU or a governance questionnaire, respectively, have been taken from related literature that does not present a comprehensive method for service identification but deals with governance aspects in general.
Activity:	Service design
Role:	SDU, employee of the IT department, (IT) manager
Technique:	Governance questionnaire, domain decomposition, naming
Result:	Naming conventions, service ownership list, modeling conventions, design principles

Fig. 4. Method Fragment 2

Based on the elements identified previously, more fragments can be designed. Elements that are used in fragments created through exploration are taken from more than one existing approach because if they were to be found in one single approach, the fragment could have been derived by decomposition as shown in section 4.1. Fragment 2 is one example for a fragment designed by exploration.

5 Conclusion and Further Research

This paper outlined the necessity of designing situation-specific methods for service identification since a literature review attested a missing configurability of existing approaches. Hence, a meta method should guide the engineering of such situational methods. An important part of this meta method is the definition of situations in the domain of service identification. Thus, the identification and discussion of context factors and SOA implementation goals was the centerpiece of this work. The idea of how to design method fragments was explained briefly and shown at two examples.

On the way to creating a meta method for the construction of situation-specific methods for service identification there are a number of limitations that should be considered. The identified context factors are based on an extensive literature research. Moreover, their significance was supported by two case studies where qualitative research methods were used. An investigation of relationships and interdependencies of these context factors and the SOAIG is subject to ongoing research.

The number of situations has to be restricted to make the approach feasible. Following [10], the context factors, their parameter values and SOA implementation goals presented herein lead to 17,280 situations. Therefore, the relevance of context factors should be scrutinized through further research. The same is true for parameter values und SOA implementation goals. The aforementioned analysis of interdependencies is also likely to reduce the number of context factor parameter values and thus the number of situations that have to be considered.

Decomposed method fragments extracted from existing methods usually have been applied to real-life projects before and are thus quite reliable. Those fragments created through exploration could be criticized for being an arbitrary combination of elements without proper foundation. Both the exemplary fragments presented in section 4 and further ones that are currently developed are based on literature and the experience of two case studies. Proving quality is difficult and indeed, completeness of a method fragment base that is proposed by many authors [45] cannot be guaranteed. Thus, it is important to feed back information from projects that will use the meta method in future. This will improve fragments and give them a stronger empirical grounding.

As demonstrated, ideas from SME can contribute significantly to the field of service identification by supporting the design of situational methods. In order to build a comprehensive meta method, experience and expert knowledge from the service-oriented domain have to be incorporated in this meta method. Further case studies or action research could support an empirical validation of this meta method.

References

1. Börner, R., Goeken, M.: Identification of Business Services - Literature Review and Lessons Learned. In: 15th AMCIS, Paper 162, San Francisco, California (2009)
2. Fitzgerald, B., Russo, N.L., O'Kane, T.: Software Development: Method Tailoring at Motorola. Communications of the ACM 46, 65–70 (2003)
3. Aydin, M.N.: Examining Key Notions for Method Adaptation. In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences, pp. 49–63. Springer, Boston (2007)

4. Harmsen, F., Brinkkemper, S., Oei, H.: Situational Method Engineering for Information System Project Approaches. In: Verrijn-Stuart, A.A., Olle, T.W. (eds.) Methods and Associated Tools for the Information Systems Life Cycle, pp. 169–194. Elsevier Science B.V., Amsterdam (1994)
5. Ralyté, J., Rolland, C.: An Approach for Method Engineering. In: 20th International Conference on Conceptual Modelling, pp. 471–484. Springer, Yokohama (2001)
6. Cossentino, M., Gaglio, S., Henderson-Sellers, B., Seidita, V.: A metamodelling-based approach for method fragment comparison. In: 11th International Workshop on Exploring Modeling Methods in Systems Analysis and Design at CAiSE, Luxembourg (2006)
7. Sunyaev, A., Hansen, M., Krcmar, H.: Method Engineering: A Formal Approach. In: 17th International Conference on Information Systems Development, Paphos, Cyprus (2008)
8. Agerfalk, P.J., Brinkkemper, S., Gonzalez-Perez, C., Henderson-Sellers, B., Karlsson, F., Kelly, S., Ralyté, J.: Modularization Constructs in Method Engineering: Towards Common Ground? In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences, pp. 359–368. Springer, Boston (2007)
9. Bucher, T., Klesse, M., Kurpjuweit, S., Winter, R.: Situational Method Engineering - On the Differentiation of "Context" and "Project Type". In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences, pp. 33–48. Springer, Boston (2007)
10. Börner, R.: Applying Situational Method Engineering to the Development of Service Identification Methods. In: 16th AMCIS, Paper 18, Lima, Peru (2010)
11. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Information & Software Technology 38, 275–280 (1996)
12. Punter, T., Lemmen, K.: The MEMA-model: towards a new approach for Method Engineering. Information & Software Technology 38, 295–305 (1996)
13. Karlsson, F., Agerfalk, P.J.: Method Configuration - Adapting to Situational Characteristics while Creating Reusable Assets. Information & Software Technology 46, 619–633 (2004)
14. Kohlborn, T., Korthaus, A., Chan, T., Rosemann, M.: Identification and Analysis of Business and Software Services - A Consolidated Approach. IEEE Transactions on Services Computing 2, 50–64 (2009)
15. Inaganti, S., Behara, G.K.: Service Identification - BPM and SOA Handshake. BPTrends 3, 1–12 (2007)
16. Goeken, M.: Entwicklung von Data-Warehouse-Systemen. Anforderungsmanagement, Modellierung, Implementierung. Deutscher Universitäts-Verlag, Wiesbaden (2006)
17. Mingers, J.: Combining IS Research Methods: Towards a Pluralist Methodology. Information Systems Research 12, 240–259 (2001)
18. Darke, P., Shanks, G., Broadbent, M.: Successfully completing case study research: combining rigour, relevance and pragmatism. Information Systems J. 8, 273–289 (1998)
19. Benbasat, I., Goldstein, D.K., Mead, M.: The case research strategy in studies of information systems. MIS Quarterly, 269–386 (1987)
20. Luthria, H., Rabhi, F.A.: Building the business case for SOA: A study of the business drivers for technology infrastructure supporting financial service institutions. In: Kundisch, D., Veit, D.J., Weitzel, T., Weinhardt, C. (eds.) FinanceCom 2008. LNBIP, vol. 23, pp. 94–107. Springer, Heidelberg (2009)
21. Yin, R.K.: Case Study Research - Design and Methods, 3rd edn., vol. 5. SAGE Publications, Thousand Oaks (2003)
22. Strauss, A., Corbin, J.M.: Basics of qualitative research: Grounded theory procedures and techniques. Sage Publications, Thousand Oaks (1990)
23. Walsham, G.: Interpretive case studies in IS research. Nature and Method. European Journal of Information Systems, 74–81 (1995)

24. Eisenhardt, K.M.: Building Theory from Case Study Research. The Academy of Management Review 14, 532–550 (1989)
25. DeLone, W.H.: Determinants Of Success For Computer Usage In Small Business. MIS Quarterly 12, 50–61 (1988)
26. Sedera, D.: Does Size Matter? Enterprise System Performance in Small, Medium and Large Organizations. In: 2nd Workshop on 3rd Generation Enterprise Resource Planning Systems, Copenhagen, Denmark (2008)
27. Welsh, J.A., White, J.F.: A amall business is not a little big business. Harvard Business Review 59, 18–32 (1981)
28. Laukkanen, S., Sarpola, S., Hallikainen, P.: Enterprise size matters: objectives and constraints of ERP adoption. J. of Enterprise Information Management 20, 319–334 (2007)
29. Becker, J., Knackstedt, R., Pfeiffer, D., Janiesch, C.: Configurative Method Engineering. In: 13th AMCIS, Paper 56, Keystone, Colorado (2007)
30. Klose, K., Knackstedt, R., Beverungen, D.: Identification of Services - A Stakeholder-Based Approach to SOA Development and Its Application in the Area of Production Planning. In: 15th ECIS, St. Gallen, Switzerland, pp. 1802–1814 (2007)
31. Terlouw, J., Terlouw, L., Jansen, S.: An Assessment Method for Selecting an SOA Delivery Strategy: Determining Influencing Factors and Their Value Weights. In: 4th International Workshop on BUSITAL, Amsterdam, The Netherlands (2009)
32. Arsanjani, A., Holley, K.: The Service Integration Maturity Model: Achieving Flexibility in the Transformation to SOA. In: IEEE International Conference on Services Computing (SCC 2006), Chicago, IL, p. 515 (2006)
33. Börner, R., Looso, S., Goeken, M.: Towards an Operationalisation of Governance and Strategy for Service Identification and Design. In: 13th IEEE International EDOC Conference, Auckland, New Zealand, pp. 180–188 (2009)
34. Leyer, M., Moormann, J.: Facilitating operational control of business services: A method for analysing and structuring customer integration. In: 21st ACIS, Paper 42, Brisbane, Australia (2010)
35. Pitts, R.A., Lei, D.: Strategic Management, 4th edn. South-Western, Mason (2005)
36. Schermerhorn, J.R.: Management for Productivity. Wiley, New York (1993)
37. Erl, T.: Service-Oriented Architecture - A Field Guide to Integrating XML and Web Services. Prentice Hall, Upper Saddle River (2004)
38. Heutschi, R.: Serviceorientierte Architektur. Springer, Heidelberg (2007)
39. Beverungen, D., Knackstedt, R., Müller, O.: Entwicklung Serviceorientierter Architekturen zur Integration von Produktion und Dienstleistung. WI 50, 220–234 (2008)
40. Papazoglou, M.P.: Service-Oriented Computing: Concepts, Characteristics and Directions. In: 4th International Conference on Web Information Systems Engineering, pp. 3–12. IEEE Computer Society, Rome (2003)
41. Kohlmann, F., Alt, R.: Deducing Service Ownerships in Financial Networks. In: 15th AMCIS, Paper 518, San Francisco, CA (2009)
42. Bieberstein, N., Bose, S., Walker, L., Lynch, A.: Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals. IBM Systems Journal 44, 691–708 (2005)
43. Henderson-Sellers, B., Ralyté, J.: Situational Method Engineering: State-of-the-Art Review. Journal of Universal Computer Science 16, 424–478 (2010)
44. Ralyté, J.: Towards Situational Methods for Information Systems Development: Engineering Reusable Method Chunks. In: International Conference on Information Systems Development (ISD 2004), Vilnius Technika, Vilnius, Lithuania, pp. 271–282 (2004)
45. Deneckère, R., Iacovelli, A., Kornyshova, E., Souveyet, C.: From Method Fragments to Method Services (2009)

An MDA Method for Service Modeling by Formalizing REA and Open-edi Business Frameworks with SBVR

Jelena Zdravkovic, Iyad Zikra, and Tharaka Ilayperuma

Department of Computer and Systems Sciences
Stockholm University and Royal Institute of Technology
Forum 100, SE-164 40 Kista, Sweden
{jelenaz,iyad,si-tsi}@dsv.su.se

Abstract. Business frameworks offer great opportunities of communication between people for working on the enterprise system engineering processes, as well as for eliciting services that the enterprise can offer in collaboration contexts. However, these kinds of frameworks, such as Resource-Event-Agent and Open-edi, recently unified in Open-edi Business Ontology (OeBTO), lack formal representations. This fact considerably limits their use in system development, particularly in model-driven development methods where the efficiency of transformations is of great importance. In this paper we suggest a formalization of OeBTO using OMG's standard Semantics of Business Vocabulary and Business Rules (SBVR), as a method for creating a service-centric business model. This makes it possible to provide the necessary formal logic foundation to allow automatic processing of the business model and its transformation to a system-level service model. An example from the bank loan business sector is used to argument the application of the method.

Keywords: business model, business collaboration, service engineering, model-driven development, REA, Open-edi, OeBTO, MDA, SBVR.

1 Introduction

Model Driven Architecture (MDA), as a formalization of Model Driven Development (MDD) approach, promotes a method for system development relying on the model transformation paradigm. MDA prescribes modeling of the business-level information as a Computational Independent Model (CIM), which is further transformed to a system-centric form called Platform Independent Model (PIM), and at the end to a Platform Specific Model (PSM) that adds the technology details needed for implementation on a specific software platform [1].

In the service-oriented business sector, capturing the consumer needs for economic resources plays an essential role in the elicitation of the services that will deliver these values, therein seizing a desirable competitive distinction. In that context, business models offer considerable advantages compared to process models - they can capture a high-level description of a whole business in a single and easily-understandable view. Business ontologies, such as Resource-Event-Agent, REA [2], facilitate modeling of actors involved in a business scenario and explain their relationships,

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 219–224, 2011.

formulating them in terms of *economic values (i.e. resources)* exchanged between the actors. Another important aspect concerns elicitation of explorative business service portfolios by spanning the whole business transaction lifecycle, which, according to the International Organization for Standardization (ISO) Open-edi initiative [3] involves planning, identification, negotiation, actualization, and post-actualization.

Recently, the ISO has focused on integrating REA and the Open-edi frameworks to create Open-edi Business Transaction Ontology (OeBTO), to specify the concepts and relationships involved in collaborative business environments. OeBTO captures the economic commitments realized by economic and business events performed by the partners, along the collaboration lifecycle in the Open-edi sense.

Following the previously outlined needs of service engineering, and MDA, in this study we consider the use of OeBTO to define a service-centric business model (i.e. CIM) and a method for its creation. To strengthen the formalism of OeBTO and thereby create an unambiguous and processable CIM, we consider the use of the OMG's Semantics of Business Vocabulary and Business Rules [4]. SBVR is an approach which allows specifying business in terms of a vocabulary and rules in a business-friendly language, while being formal enough to be readable by systems.

Being rooted in the use of MDA and two well-established business frameworks; REA and Open-edi formalized by SBVR, we believe that the method we propose for creating CIM forms a solid basis to be efficiently transformed to a SOA-aligned system model and further to Web services.

The rest of the paper is organized as follows. Section 2 gives the overviews on the used business frameworks, and SBVR. In Section 3 we present our method for identification and modeling of business services. In Section 4, we refer to the work related to ours, and conclude the paper.

2 Background

In this section, we briefly describe REA and Open-edi business frameworks and their integration in Open-edi Transaction Ontology (OeBTO); then we give an overview of the Semantics of Business Vocabulary and Business Rules (SBVR) standard.

Integration of REA and Open-edi Business Frameworks in OeBTO

The core concepts in the Resource-Event-Agent (REA) framework are *resource*, *event*, and *agent* [2]. It is assumed that every business activity can be described as an event where two agents exchange economic values, i.e. resources. Economic resources may be classified as *goods*, *rights* or *services*. To acquire a resource, an agent (i.e. actor) has to give up some other resource (*economic duality*). In the study [5], the REA framework has been extended to capture additional granularity levels of business activities of enterprises. The resulting framework has integrated three vertical layers: *Value Chain, Business Process* and *Business Event*:

Open-edi Business Transaction Ontology (OeBTO) extends the REA ontology with the concepts aimed to facilitate the modeling of business collaborations defined in the ISO Open-edi initiative [6]. According to Open-edi, business collaborations span five phases: *planning, identification, negotiation, actualization* and *post-actualization*. In *the planning phase*, the customer and the provider are engaged in

activities to identify the actions needed for selling or purchasing goods and services. *The identification phase* involves the activities needed to exchange information among providers and potential customers regarding selling or purchasing goods and services. During *the negotiation phase*, contract terms are proposed and completed. *The actualization phase* includes all the activities necessary for exchanging goods and services between involved actors as agreed during negotiations. The *post-actualization phase* encompasses the activities and associated exchanges between involved actors after the major resources are provided.

SBVR

Recently, the Object Management Group (OMG) has adopted the Semantics of Business Vocabulary and Business Rules (SBVR) as a standard for capturing business vocabularies and rules [4]. The term "business" in SBVR is used in a general sense, referring not only to activities that imply an exchange of goods or services for money, but also to other types of activities where rules need to be defined and documented, such as education, health care, or law [7].

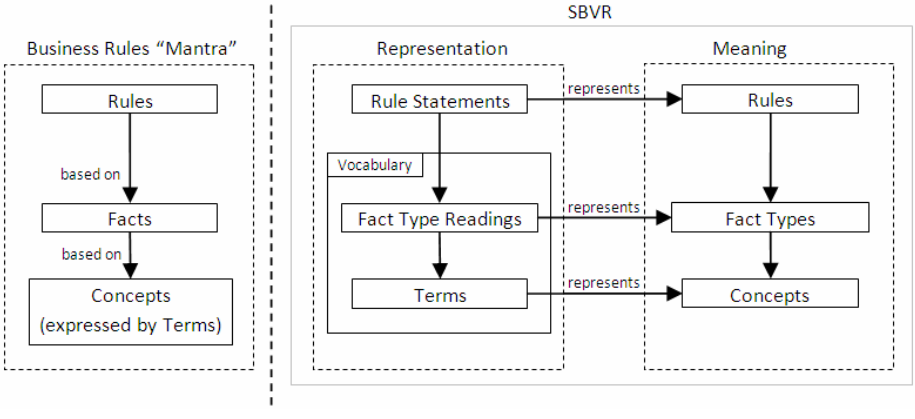


Fig. 1. Meaning, representation, and the support of business rules in SBVR

The basic idea around which SBVR is developed is that business rules are built on top of facts, which in turn are built on top of terms that represent concepts. Furthermore, SBVR acknowledges the difference between meaning and the representation used to convey that meaning. Figure 1 above shows how SBVR realizes the business rules and also emphasizes the independence of meaning and its representation.

3 Creating Service-Centric SBVR-Based Business Model

In this study we utilize the MDA method to model services. In that effort, we consider REA and Open-edi (i.e. OeBTO) as an established conceptual basis for the business collaboration context. To obtain a service-centric CIM, we propose the following:

– A classification of the notion of Business Transaction in OeBTO to enable CIM to describe different value configurations.

– An extension to the original OeBTO to capture service-related notions, such as structure, behavior or policies.
– A (re)formulation of OeBTO using SBVR, to increase the formalism of CIM and thereby facilitate transformations to the PIM level.

Method for Creating a Three-layered Business Model

According to the REA framework, the method for describing the business of an enterprise comprises the decomposition of business activities along three granularity layers: value chain, business processes and business events (Figure 2). In the following, we will outline a method for creating each of the layers, where the concepts on each of the layers are defined using SBVR-based OeBTO.

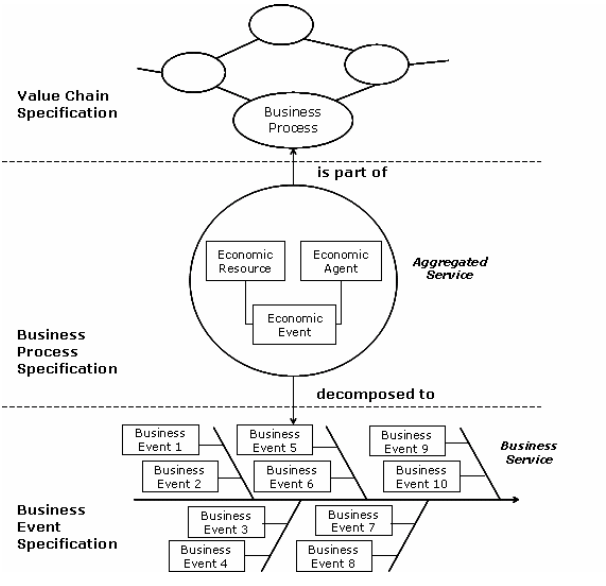


Fig. 2. The three-layered REA business framework

Step 1: Value Chain Specification

In this step the business (i.e. value-adding) processes in the highest layer of the REA framework are identified, using a service-aware value configuration, such as the classification that includes traditional *value chain, value shop,* and *value network* [8].

Step 2: Business Process Specification

Moving to the middle layer of the REA framework, each identified business process (transaction, in OeBTO) is explored to find the partners involved in it, as well as the economic resources being exchanged. Each economic exchange gives rise to an *aggregated service*, which will be further expanded on the next layer of the framework, to discover the actual business services that will realize the delivery of the economic exchange.

Step 3: Business Event Specification
At the bottom layer of the REA framework, the economic events of the economic exchange are expanded over the five Open-edi business transaction phases. This is intended to discover the candidate business services and business events. According to OeBTO, the business event is used to represent the business activities elicited for every business transaction phase at the third layer. In order to capture the services that compose the aggregated service and at the same time aggregate related business events, we introduce the *business service* element. A business service is a standalone service that can be reused as part of other aggregated services to provide other economic exchanges. In Figure 3, a small excerpt of the formalization of OeBTO using SBVR is illustrated. For the full specification, the reader is referred to [9]

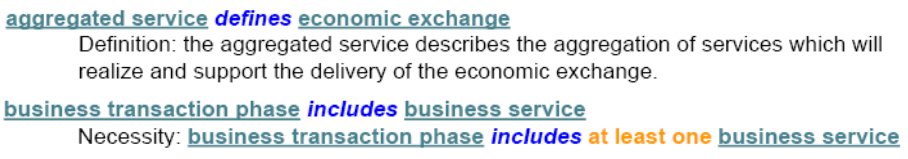


Fig. 3. An excerpt of the formalization of OeBTO using SBVR

From the MDA perspective, the obtained service-aware OeBTO model is used as the input for creating a system model, i.e. Platform Independent Model (PIM). Since SBVR is completely grounded in formal logic, it gives the added benefit of automatic model processing, i.e.:

- Making it possible for tools to ensure the integrity of OeBTO models.
- Ensuring the integrity of transformations that produce PIM models based on the OeBTO model.

5 Related Work and Conclusion

In this study, we have applied the MDA method to design business-services, which may be further transformed to system services and implemented using Web services.

Lately, research in both academic and industrial communities have implied that when designing service-oriented software solutions, the starting point should be the business models of enterprises [10], [11], [12] and [13]. This fact, according to the referred studies, is shifting the focus of large scale e-service design to the context of economic resource transfers. Our method reported in [14] differs from those studies in the way that we set the focus on the analysis of business transactions relying on the OeBTO standardization effort, and expanding them along a number of collaboration phases to get a rich business service portfolio. The aim of this study has been to further improve the use of OeBTO for service modeling in the MDA method, by formalizing it with SBVR, and to use SBVR to guide transformations toward PIM.

The major strength of the proposed method is the use of REA and Open-edi frameworks formalized with SBVR for modeling CIM; in that way we have obtained a method which facilitate creating a declarative-type CIM, unambiguous and

processable, i.e. with the capability to be further transformed with a high extent of automation to system-centric service model (PIM).

References

1. Kleppe, A., Warmer, J., Bast, W.: MDA Explained. Addison-Wesley Professional, Reading (2003)
2. McCarthy, W.E.: The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. The Accounting Review (1982)
3. ISO/IEC: Operational aspects of Open-edi implementation. ISO Standard 15944-1 (2002)
4. The Object Management Group (OMG): Semantic of Business Vocabulary and Business Rules (SBVR), v1.0. OMG standard, http://www.omg.org/spec/SBVR/1.0/PDF
5. Geerts, G., McCarthy, W.E.: An Ontological Analysis of the Primitives of the Extended-REA Enterprise Information Architecture. The International Journal of Accounting Information Systems 3, 1–16 (2002)
6. ISO/IEC: Business transaction scenarios - Accounting and economic ontology. ISO Standard 15944-4 (2007)
7. Linehan, M.H.: SBVR Use Cases. In: Bassiliades, N., Governatori, G., Paschke, A. (eds.) RuleML 2008. LNCS, vol. 5321, pp. 182–196. Springer, Heidelberg (2008)
8. Stabell, C.B., Fjeldstad, O.D.: Configuring value for competitive advantage: on chains, shops, and networks. Strategic Management Journal 19, 413–437 (1998)
9. Zikra, I.: SBVR-based Service-extended OeBTO Meta Model. Technical Report, http://people.dsv.su.se/~iyad/public/SBVR-Based_OeBTO_Meta_Model.pdf
10. Baida, Z., Gordijn, J., Saele, H., Akkermans, H., Morch, A.: An Ontological Approach for Eliciting and Understanding Needs in e-Services. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 400–414. Springer, Heidelberg (2005)
11. Cherbakov, L., Galambos, G., Harishankar, R., Kalyana, S., Rackham, G.: Impact of Service Orientation at the Business Level. IBM Systems Journal 44(4), 653–668 (2005)
12. Anderssson, B., Johannesson, P., Zdravkovic, J.: Aligning Goals and Services through Goal and Business Modeling. The International Journal of Information Systems and e-Business Management (ISEB), Special Issue on Design and Management of Business Models and Processes in Services Science 7, 143–169 (2007)
13. Gordijn, J., van Eck, P., Wieringa, R.: Requirements Engineering Techniques for e-Services. In: Georgakopoulos, D., Papazoglou, M.P. (eds.) Service-Oriented Computing: Cooperative Information Systems Series, pp. 331–352. The MIT Press, Cambridge (2009)
14. Zdravkovic, J., Ilayperuma, T.: A Model-driven Approach for Designing E-Services Using Business Ontological Frameworks. In: Proceedings of 14th IEEE International EDOC Conference, pp. 121–130. IEEE Computer Society, Los Alamitos (2010)

A Scenario-Based Governance Method for Coordination of Service Life Cycles

Sietse Overbeek, Marijn Janssen, and Yao-Hua Tan

Faculty of Technology, Policy and Management,
Delft University of Technology,
Jaffalaan 5, 2628 BX Delft, The Netherlands, EU
{S.J.Overbeek,M.F.W.H.A.Janssen,Y.Tan}@tudelft.nl

Abstract. Most of today's organizations are still far from profiting from the full potential of web service technology. Organizations invoke each other's web services, but do hardly synchronize the life cycles of individual web services with each other. When realizing an integrated service this might cause failures and requires a governance method for synchronizing the life cycles among organizations. In this paper, we emphasize that there is a need for such a method and the basis of the method is explained. This consists of the identification of possible life cycle related scenarios when trying to integrate services. The method also provides support to coordinate and track changes in service life cycles. Based on the different scenarios when attempting to integrate services that have different life cycles, coordination to communicate life cycle changes and having clear agreements about expectations are needed for successful service integration.

1 Introduction

Web service technology is becoming the solution to make the business services of single organizations and inter-organizational coalitions available online and to match the supply of services and the demand for services by clients (see e.g. [5]). A web service can be defined as a software component identified by a URI, whose interfaces and bindings can be defined, described and discovered as XML artifacts [3]. However, most of today's organizations are still far from profiting from the full potential of web service technology consisting of the computerization, integration and matching of services. In fact, it is common practice for most organizations to develop their web services by adding a thin SOAP / WSDL / UDDI layer on top of existing software applications or components [5]. While simple services may be constructed this way, it is not sufficient for realizing and offering a dynamically created integrated web service that matches complex and varying client needs. The notion of a *service-oriented life cycle methodology* has been introduced to design, implement, monitor, and manage web services in such a way that organizations can benefit in full from the advantages of web services. Such methodologies also provide sufficient principles and guidelines to specify, construct, refine and customize highly flexible business processes taken from a

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 225–230, 2011.

set of internal and external web services [4]. This enables that the specification and execution of business processes is aligned with those business services that are transformed to web services. Examples of existing methodologies can be found in [1,4]. In practice, organizations invoke each other's services and become dependent of those services. Despite these dependencies, organizations adopt or create their own methodologies which are often unrelated to those of other organizations. This increases the risk that the various life cycles are out of sync which might result in failures.

This is a complicating factor when integrating web services and the accompanying business processes that need to be executed to supply the integrated service. In the context of supply chain logistics, for example, an integrated web service can be supplied to a client who wishes to declare veterinary cargo online and to track and trace that type of cargo. If the integrated web service is a new web service that is going to be offered by two different organizations then their life cycle methodologies should be applied synchronously. This will prevent, e.g., communication problems and delays in the joint development of the integrated service and the accompanying cross-organizational business process. Mismatches between life cycles can also occur if, for example, an integrated web service needs to contain both existing and new web services. A service that already exists is in a different phase of its life cycle than a newly created service. Moreover, organizations that collaborate regularly innovate their processes, methods and business models and in this way they are in need for substituting old services with new ones. In [6], the need for these innovations are also linked to those organizations that rapidly expand. Innovation causes organizations to phase out old services and add new services. This implies again that these services are in different phases of their life cycles. In this paper, we present a governance method for supporting the coordination of web service life cycles. The aim of the method is to ensure that the dependencies among web services from different organizations are managed during the complete life cycle. This scenario-based method provides a way of working for service providers in coordinating and keeping track of life cycle changes. Section 2 introduces four different scenarios that can occur when attempting to integrate web services that have different life cycles. Subsequently, the basis for the proposed governance method is presented in section 3. Finally, the conclusions of this paper are presented in section 4.

2 Scenarios for Comparison of Service Life Cycles

In order to understand how web services and their life cycles can be compared and to determine to what extent they match, we formalize four possible matching scenarios. A life cycle methodology is typically divided in various phases to depict in which position a web service is in its life cycle. The phase equation is used to determine which phases uniquely belong to which life cycle methodology: $\mathsf{Phase} : \mathcal{PS} \rightarrow \mathcal{LC}$. If a phase $p \in \mathcal{PS}$ is part of a life cycle methodology $l \in \mathcal{LC}$, this can be expressed as $\mathsf{Phase}(p) = l$. In this case, the set $\mathcal{PS}$ is the set of phases and $\mathcal{LC}$ is the set of service-oriented life cycle methodologies. The phase

classification equation is used to determine in which phase of its life cycle a web service is classified: $\mathsf{Classification} : \mathcal{SC} \rightarrow \mathcal{PS}$. For example, if a service $s \in \mathcal{SC}$ is in phase $p \in \mathcal{PS}$ of its life cycle, this can be expressed as $\mathsf{Classification}(s) = p$. In this case, the set $\mathcal{SC}$ is the set of services. When different life cycle methodologies are used, it may be impossible to compare two web services that have life cycles that are based on different methodologies. This is because different terms to describe a phase are used and web services can be classified in a certain phase based on different criteria. However, comparison may still be possible if some phases of each life cycle are semantically similar. For example, a 'planning' phase in one life cycle may have the same meaning as an 'initiation' phase in another life cycle. Semantic similarity between phases of life cycle methodologies is modeled as follows: $\mathsf{Similarity} : \mathcal{PS} \times \mathcal{PS} \rightarrow [0, 1]$. The lack of semantic similarity between two phases $p_1, p_2 \in \mathcal{PS}$ can be expressed as $\mathsf{Similarity}(p_1, p_2) = 0$, while the opposite result is true for full semantic similarity between two different phases. Four scenarios are imaginable when matching life cycles: (1) Two different services are in the same phase of their life cycles and the life cycles are also applications of the same methodology. (2) Two different services are in different phases of their life cycles and the life cycles are again applications of the same methodology. (3) Two different services are in different, but semantically similar phases of their life cycles that are based on two different methodologies. (4) Two different services are in semantically distinct phases of life cycles that are based on two different methodologies. These scenarios can be described more formally by using the above equations:

$$\exists_{l \in \mathcal{LC}} \exists_{p \in \mathcal{PS}} \exists_{s_1, s_2 \in \mathcal{SC}} [\mathsf{Phase}(p) = l \wedge \mathsf{Classification}(s_1) = p \wedge \mathsf{Classification}(s_2) = p] \quad (1)$$

$$\exists_{l \in \mathcal{LC}} \exists_{s_1, s_2 \in \mathcal{SC}} [\mathsf{Phase}(\mathsf{Classification}(s_1)) = l \wedge \mathsf{Phase}(\mathsf{Classification}(s_2)) = l \wedge \mathsf{Classification}(s_1) \neq \mathsf{Classification}(s_2)] \quad (2)$$

$$\exists_{l_1, l_2 \in \mathcal{LC}} \exists_{s_1, s_2 \in \mathcal{SC}} [\mathsf{Phase}(\mathsf{Classification}(s_1)) = l_1 \wedge \mathsf{Phase}(\mathsf{Classification}(s_2)) = l_2 \wedge \mathsf{Classification}(s_1) \neq \mathsf{Classification}(s_2) \wedge \mathsf{Similarity}(\mathsf{Classification}(s_1), \mathsf{Classification}(s_2)) = 1] \quad (3)$$

$$\exists_{l_1, l_2 \in \mathcal{LC}} \exists_{s_1, s_2 \in \mathcal{SC}} [\mathsf{Phase}(\mathsf{Classification}(s_1)) = l_1 \wedge \mathsf{Phase}(\mathsf{Classification}(s_2)) = l_2 \wedge \mathsf{Classification}(s_1) \neq \mathsf{Classification}(s_2) \wedge \mathsf{Similarity}(\mathsf{Classification}(s_1), \mathsf{Classification}(s_2)) = 0] \quad (4)$$

Under normal circumstances, it can be expected that in the first matching scenario the least difficulties exist to integrate web services and that these difficulties will gradually increase up until the fourth scenario. If this assumption is true, this would mean that there is a causal relation between the life cycles of web services and the ability to integrate and supply them as one integrated service. Examples of causes for integration difficulties are: disabilities to comprehend service designs, conflicting service designs, temporal differences between activities performed in comparable phases and different service maintenance levels.

3 Scenario-Based Governance Method for Life Cycle Coordination

The proposed governance method is based on the underlying thought that *changes* in the life cycles of web services require proper coordination, just like the matching of supply and demand of services should be coordinated [2]. This will also include the final agreements on these changes by the different owners of the web services. Changes in life cycles need to be announced and the time-line to make actual changes to the services need to be agreed on. A governance method that can be used by service providers as a *way of working* to keep track of desired changes and to coordinate them to ensure the proper functioning of an integrated service is then called for. The governance method consists of two parts, which concerns the activity diagrams shown in figures 1 and 2. The first diagram shows how to determine which of the four presented scenarios apply when attempting to realize an integrated service. To determine which scenario can be considered, the life cycle methodologies of the services that would be part of a new integrated service are compared. If the methodologies are identical, the phases of

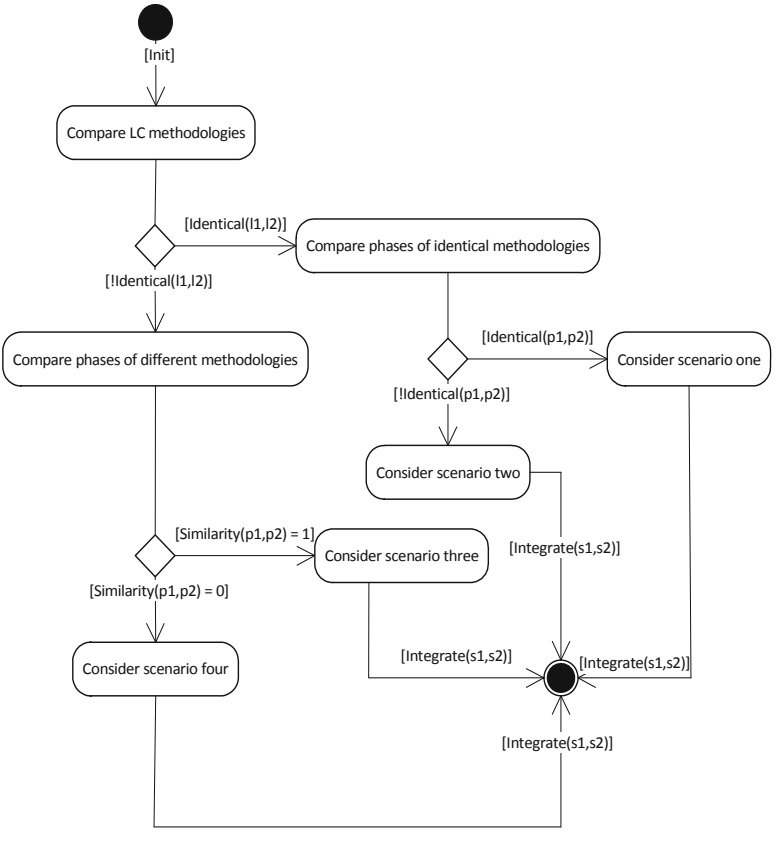


Fig. 1. Determining life cycle scenario when realizing an integrated service

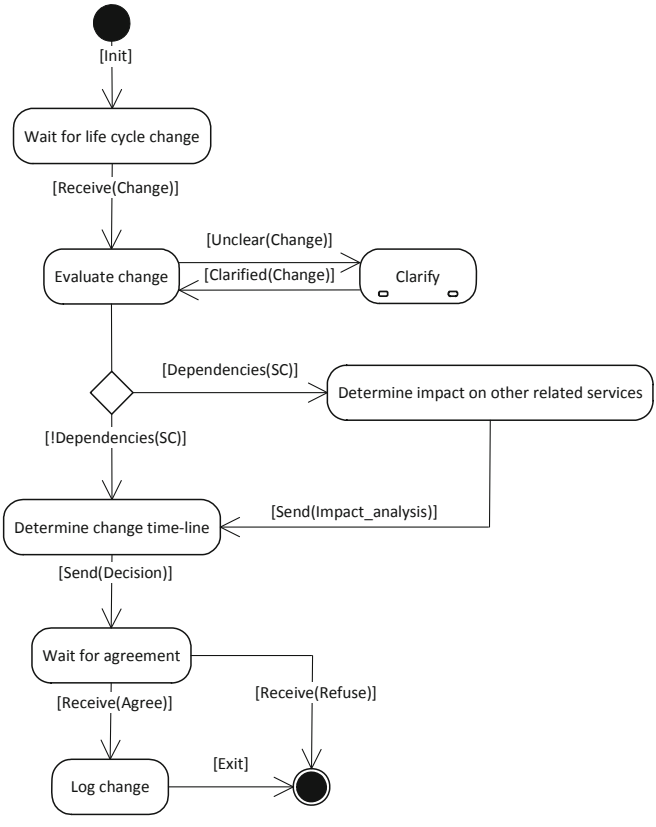


Fig. 2. Coordination and tracking of changes in service life cycles

the service life cycles can be compared. If those phases are also identical, scenario one can be considered. If not, the life cycles resemble scenario two. If the methodologies are not identical, it should be determined whether the life cycle phases are semantically similar or not. If this is the case, we arrive at scenario three. If it is not the case, we arrive at scenario four. The integration procedure can be started after determining with which scenario is dealt with and taking into account the possible issues that can arise related to that scenario. When the proper scenario is determined, service providers are more aware of which possible issues they may face during the integration process. The second diagram shows how to coordinate and keep track of changes in service life cycles. A change in a life cycle of a service that is known to a service provider will be evaluated after its reception. More clarification concerning the change can be requested if this is needed. The clarify state represents a composite state, which is not shown further. The impacts of the change on other services are determined if there are dependencies between the service of which the life cycle has changed with other services. Next, the time-line of the change is determined. After determining this, the change can be either agreed or refused. The change is logged in a registry

of life cycle changes if it is agreed upon. Both the scenario determination part and the part to coordinate life cycle changes can be used by service providers to control attempts to realize integrated services.

4 Conclusions

The results of the presented research show the basis of a governance method for the coordination of web service life cycles. The motivation to create such a method is rooted in the observation that organizations invoke each other's web services when realizing an integrated service for their clients, but that the life cycles of those services are hardly synchronized. The actual governance method consists of two parts for supporting the coordination of life cycles. One part shows how to determine with which life cycle scenario a service provider is confronted when integrating services. The second part provides a way of working for service providers in coordinating and keeping track of life cycle changes. By adopting the governance method, coordination to communicate changes and having clear agreements about expectations is then realized based on the different scenarios when attempting to integrate services that have different life cycles.

References

1. Bianchini, D., Cappiello, C., De Antonellis, V., Pernici, B.: P2S: A methodology to enable inter-organizational process design through web services. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 334–348. Springer, Heidelberg (2009)
2. Burstein, M., Bussler, C., Zaremba, M., Finin, T., Huhns, M., Paolucci, M., Sheth, A., Williams, S.: A semantic web services architecture. IEEE Internet Computing 9(5), 72–81 (2005)
3. Ferris, C., Farrell, J.: What are web services? Communications of the ACM 46(6), 31–34 (2003)
4. Papazoglou, M., van den Heuvel, W.J.: Service-oriented design and development methodology. International Journal of Web Engineering and Technology 2(4), 412–442 (2006)
5. Papazoglou, M., van den Heuvel, W.J.: Business process development life cycle methodology. Communications of the ACM 50(10), 79–85 (2007)
6. van de Weerd, I., Brinkkemper, S., Versendaal, J.: Incremental method evolution in global software product management: A retrospective case study. Information and Software Technology 52(7), 720–732 (2010)

Author Index