Valeria de Castro, Juan Manuel Vara, Esperanza Marcos,
Mike Papazoglou, Willem-Jan Van den Heuvel (Eds.)

2nd International Workshop on Model-Driven Service Engineering (MoSE 2010)

Málaga, Spain, 29 June 2010.

MoSE 2010

Proceedings of the 2nd International Workshop on Model-Driven Service Engineering

In conjunction with the

TOOLS 2010 Federated Conferences

Organized by

Kybele Research Group
Department of Languages and Computing Systems II
Rey Juan Carlos University

Supported by

Rey Juan Carlos University

Preface

Model-Driven Engineering (MDE) deals with the provision of models, transformations between them and code generators to address software development. One of the main advantages of model-driven approaches is the provision of a conceptual structure where the models used by business managers and analysts can be traced towards more detailed models used by software developers. This kind of alignment between high level business specifications and the lower level Service Oriented Architectures (SOA) is a crucial aspect in the field of Service-Oriented Development (SOD) where meaningful business services and business process specifications are those that can give support to real business environment usually changing with increasing speed.

SOD has become currently in one of the major research topics in the field of software engineering, leading the appearance of a novel and emerging discipline called Service Engineering (SE), which aim to bring together benefits of SOA and Business Process Management (BPM). SE focuses on the identification of service (a client-provider interaction that creates value for the client) as first class elements for the software construction. The convergence of SE with MDE can holds out the promise of rapid and accurate development of software that serves software users' goals.

In this context, the *2nd Workshop on Model Driven Service Engineering (MoSE 2010)* aims to provide a forum to discuss different issues related to SE in conjunction with MDE, boarding open research problems in this area as well as practical experiences. Particular interests include methods, modelling languages, development methodologies and techniques in the field of SOD.

We have received in this edition 10 contributions. All the papers received have been reviewed by, at least, three members of the international Program Committee. As the result of the review process 5 works were accepted as regular papers for their presentation at MoSE 2010 workshop. Moreover, we have also heard Dr. Eelco Visser from the Department of Software Technology at Delft University of Technology who presented the invited lecture: "Service Models for WebDSL and Mobl".

We wish to thank all the contributors to MoSE 2010, in particular the authors who submitted papers and likewise, we acknowledge the time and effort

contributed by all the members of the Program Committee who have very carefully reviewed the submitted papers. In closing, we would like to thank the Rey Juan Carlos University for their financial support.

June 2010

Valeria De Castro
Juan Manuel Vara
Esperanza Marcos
Mike Papazoglou
Willem-Jan Van den Heuvel
Organization Chair
MoSE 2010

Workshop Organization

Workshop Organizers

Valeria De Castro	Rey Juan Carlos University, Spain
Juan Manuel Vara	Rey Juan Carlos University, Spain
Esperanza Marcos	Rey Juan Carlos University, Spain
Mike Papazoglou	Tilburg University, Netherlands
Willem-Jan Van den Heuvel	Tilburg University, Netherlands

Progam Committee

Yuan An	Drexel University, USA
Rolv Braek	University of Science and Technology, Norway
Jorge Cardoso	University of Coimbra, Portugal
Alfonso Castro	Telefonica I+D, Spain
Rafael Corchuelo	University of Seville, Spain
Marcos Didonet Del Fabro	IBM Software Group, France
Ruben Fuentes	Technical University of Madrid
Nora Koch	Ludwig Maximilians University, Germany
Guadalupe Ortiz Bellot	University of Cadiz, Spain
Genoveva Vargas Solar	CNRS, LSR-IMAG, France

Organizing Committee

Verónica Bollati	Rey Juan Carlos University, Spain
Carlos Cuesta	Rey Juan Carlos University, Spain
Elisa Herrmann	Rey Juan Carlos University, Spain
Marcos López	Rey Juan Carlos University, Spain
Diana Sánchez	Rey Juan Carlos University, Spain
Belén Vela	Rey Juan Carlos University, Spain

Table of Contents

Modelling Self-Management Requirements in Service-Oriented Systems using SelfMML

Carlos Rodríguez-Fernández[1], Jorge J. Gómez-Sanz[1], and Juan Pavón[1]

Facultad de Informática,
C/ Prof. José García Santesmases, s/n,
28040 Madrid, Spain
{carlosro,jjgomez,jpavon}@fdi.ucm.es

Abstract. This paper introduces a language called Self-Management Modelling Language (SelfMML) which supports the modelling of self-management capability requirements. The paper presents a case study related to the automatic re-binding features in services which illustrates and analyses the language usage in service-oriented systems.

1 Introduction

Software Systems intended to provide services usually should be operative at peak performance 24/7 to meet the end-user needs and the business requirements. Therefore, it is desired that such systems have the capability of managing themselves without human intervention, since a system could done such operations in a faster and more accurate way. This capability is called self-management[1].

The self-management definition is detailed by its four aspects: self-optimisation, self-configuration, self-healing and self-protection[1]. *Self-optimisation* is the automatic looking and finding of opportunities to tune the system to improve the performance and the efficiency; *Self-configuration* is the automatic configuring of the system following high level policies; *Self-healing* is the automatic recovering from unhealthy state; and *Self-protection* is the protecting itself from attacks and cascading errors, with anticipatory or reactive actions [1].

Obtaining such capabilities in a service-oriented system is not a trivial work and require a costly engineering effort. This work could be facilitated if the developer had specialised tools which support the definition of those capabilities. To support this claim, this paper studies the impact of applying the Self-Management Modelling Language (SelfMML) for the modelling of self-management capability requirements in a service-oriented system.

The Self-Management Modelling Language (SelfMML) is a language which intends to assist in the engineering of self-management capability requirements, providing visual representations related to the specification of them. A visual editor tool for this language is provided to create, view, edit and store SelfMML specifications. This tool can be downloaded at `http://selfmml.sf.net`.

The chosen case study for applying SelfMML is based on a well known scenario in service-oriented computing: the re-binding. Specifically, the re-binding

capability requirement in an on-line blog system, which is considered as a self-management capability requirement that will be modelled using the proposed language.

It is important to remark that we focus on the modelling aspects, not in the requirement engineering process. Hence, we intend to provide a modelling language that permits a developer to effectively capture and specify a self-management requirement, but we do not provide assistance for the requirement engineering process that uses this language. The SelfMML scope is limited to the self-management requirement specification. A self-management capability represents an expected behaviour from the system, which should be implemented in some way during the chosen development process. During the realisation of these, the engineer can identify elements and design the architecture of the final system according to the specified expected behaviour, analogous to the realisation of use cases. The developer could consider some framework or reference architecture for self-management or just develop an ad-hoc solution. A method to realise and verify self-management requirements modelled by SelfMML is not study in this work.

This work is structured as follows. Section 2 describes the language SelfMML. Section 3 presents a case study used as illustrative example of the language usage. Section 4 shows some related work that has been taken into account for this language. Section 5 introduces the conclusions.

2 The Self-Management Modelling Language (SelfMML)

The Self-Management Modelling Language (SelfMML) is a language to be used in the modelling of self-management capabilities that a system should have, that is, self-management capability requirements. This language is made from UML 2.2 Superstructure, concretely, copies all elements from the *Use Cases* and *Activities* packages and extends them with new elements. Also imports elements from the *Kernel* package for the definition of the elements.

The language has a meta-model that defines its abstract syntax. Further information about the meta-model can be found in `http://selfmml.sf.net`. The language is described below (see figures 1, 2 and 3).

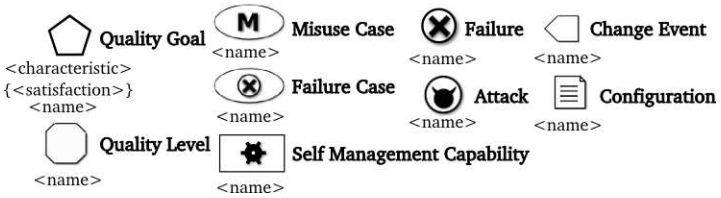

Fig. 1. SelfMML (1)

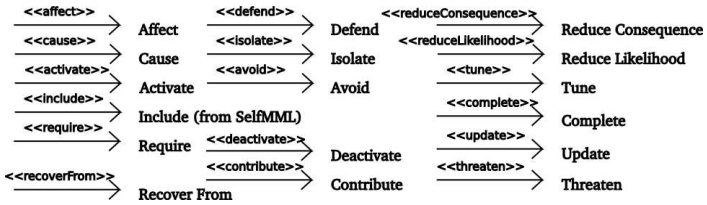

Fig. 2. SelfMML (2)

– **Self-Management Capability**. Self-Management Capabilities are the abilities of systems to do management operation by themselves on themselves. This element is provided for the representation of self-management capability requirements in a system.

Self-Management Capabilities are usually related to quality requirements as maintainability, portability, reliability, usability, availability, among others [2,3], in a way that contribute to the satisfaction of them. The language has the Quality Goal and Quality Level elements for the representation of quality requirements (see figure 1).

– **Quality Goal**. This element represents a quality requirement described as a goal that the system should maintain. It has a *characteristic* property that describes the characteristic or factor related to the quality requirements, and also has a *satisfaction* property that describes how the quality goal is satisfied by the using of some expression in natural language or another language such as Object Constraint Language (OCL). Such expressions could describe what are acceptable values for the quality metrics related to the quality requirement, following the IEEE 1062-1998 Standard recommendations.
– **Quality Level**. This element represents a quality level which groups quality goals that the system should maintain.

The language lets developers model how self-management capabilities are related to Quality Requirements and Use Cases, by the using of the relationships *contribute* for quality goals and *require* for use cases. Also the *include* relationship is provided to describe what quality goals are included in a specific quality level (see figure 2). Quality Goals can be connected to describe contribution with the *contribute* relationship too.

Problems are the target of self-protection and self-healing capabilities. The language provides elements to model possible problems in the system following the philosophy from the Failure Modes and Effect Analysis methods (FMEA)[4]. These elements are:

– **Failure**. This element describes a failure that could happen in the system.
– **Failure Case**. This element describes what is the wrong behaviour that a system shows when some failure has happened (failure modes in FMEA).
– **Misuse Case**. This element describes a misuse of a system that an user does which can lead to a failure[5,6,7].

– **Attack**. This element describes an attack against the system.

A failure, a misuse case or an attack can be related to a quality goal with the *threaten* relationship indicating that the first ones threaten the satisfaction of the second ones. Failures can be connected to failure cases with the *activate* relationship indicating that the first ones activate the wrong behaviour described in the second ones. Also failures can be connected to use cases with the *affect* relationship indicating that the first ones affect the normal operation of the functionality described in the second ones. Failures, misuse cases and attacks can be connected to failures with the *cause* relationship indicating that the first ones cause the second ones.

A self-management capability, as a self-protection capability, can be connected to problems (failures, misuse case and attacks) to indicate that the capability takes anticipatory actions to avoid, reduce the consequence or reduce the likelihood of such problems using the *avoid*, *reduceConsequence* and *reduceLikelihood* relationships respectively. Also, as a self-protection capability, the elements can be connected to attacks with the *defend* relationship to indicate that the system reacts to such attacks; and can be related to failures with the *isolate* relationship to indicate that the system isolates such failures to avoid the cascading failures that they could cause.

A self-management capability, as a self-healing capability, can be connected to a failure with the *recoverFrom* relationship indicating that the capability recovers the system from the failure. Also, it can be connected to a failure case with the *deactivate* relationship indicating that the capability deactivates the wrong behaviour of the system.

According to the self-configuration and self-optimising aspect of the self-management, a self-management capability could have the intention to tune a certain configuration parameters to improve the performance of the system, and could also have the intention to complete or update a certain configuration parameters following high level policies, in reaction to change events in the system or in the environment. SelfMML provides two elements to model such configurations and change events:

– **Change Event**. This element describes a change event in a system.
– **Configuration**. This element describe a configuration description.

A self-management capability could be related to a configuration with the following relationships: *tune*, *complete* and *update*, indicating that the capability try to tune, compete or update the configuration. A capability could be related to a change event with *treat* relationship indicating that the capability treats the event.

SelfMML provides a set of activity nodes to specify the abstract process of a self-management capability using activities diagrams (see figure 3). A self-management process usually has a structure according to four phases[1]: monitoring, analysis, planning, and plan execution. In monitoring phase the interesting information is gathered; in the analysis phase, the information is processed in order to infer some needed knowledge; in the planning phase the obtained

knowledge is used to decide what plan executes or to build a new one; finally in the plan execution phase the selected or built plan is executed.

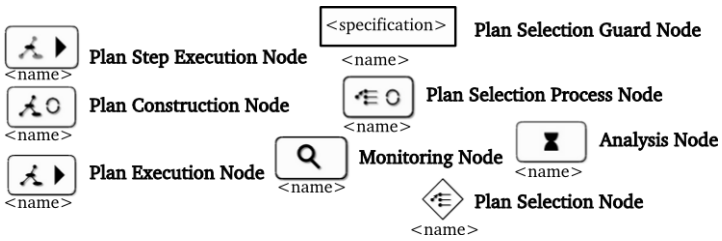

Fig. 3. SelfMML (3)

The elements for monitoring and analysis are the following (see figure 3):

- **Monitoring Node**. This element describes the monitoring activities and it continually generates tokens after each monitoring cycle. The method to obtain information by monitoring could be pulling, pushing, interception or any other kind or variant; the node does not imply the using of any particular method.
- **Analysis Node**. This element describes the analysis activity to infer knowledge from the monitoring reports.

The provided elements to model the selection or construction of plans are the following (see figure 3):

- **Plan Selection Node**. This element is for the selection of plans using OCL expression, it receives tokens from the incoming edge and copy one for each outgoing edges. The continuity of the token depends on the *Plan Selection Guard Node*.
- **Plan Selection Guard Node**. This element is to constrain the execution of plans, it receives tokens from the incoming edge and presents them to the outgoing edges only if the OCL expression described in the *specification* property is evaluated to true.
- **Plan Selection Process Node**. This element describes a selection which is done by a more sophisticated process that cannot be expressed by OCL. It copies all incoming tokens to all outgoing edges, but the continuity of such tokens depends on the evaluation of the guards on the outgoing edges. Then, a simple terms in guards can be used to describe what plan is selected.
- **Plan Construction Node**. This element describes a plan construction activity.

In order to specify the plan execution activities the language provides the following elements (see figure 3):

- **Plan Step Execution Node**. This element describes a step of a certain plan to be executed.

– **Plan Execution Node**. This element describes an undefined plan to be executed (useful when the plan is constructed).

3 Case Study: The Service Re-binding

The studied system is a blog system such as *blogger.com*. Publisher can submit posts using a publishing service. This publishing service lets users write a post and attach to the post any kind of files. It will use an external storage service to store the attached files.

This service is constrained by two quality requirements related to the availability and the reliability. Both requirements are included in an acceptable quality level for standard users. The first requirement constrains the availability of the publishing service in a value that should be equal or greater than 98%. The second requirement constrains the reliability in a fault response likelihood value equal or less than 0.05 (see figure 4).

Several problems can affect the requirements fulfilment, this paper identified only four of them. There are two failures that can affect the normal operation of the service: the used storage service becomes unavailable; and the used storage service gives too many fault responses, becoming unreliable. Both failures can cause other two: the unavailability and the unreliability of the publishing service respectively, and can threaten the quality requirements satisfaction described before (see figure 4).

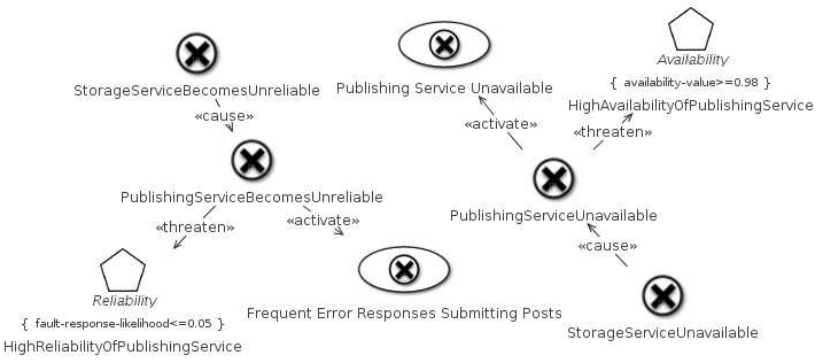

Fig. 4. Failures and Quality Goals diagram

Both failures in the publishing service activate two failure cases (see figure 4). The *Frequent Error Responses Submitting Posts* describes that "randomly" the system answers with an unexpected error when the publisher is submitting a new post. The *Publishing Service Unavailable* describes that when the publisher accesses to the service the system shows the message "The Publishing Service is Unavailable, please try later".

In order to treat these problems, there is a requirement: "The system should have the capability of detecting such problems and automatically re-bind to an alternative Storage Service, following a given selection criteria based on the quality levels offered by them. It assumes that exist a Registry that have a full description of Storage Services that can be used by the Publishing Service. The system should try to select the service that offers the highest quality level related to the availability and the reliability. But, when the selection is not clear (because exist an alternative with the highest availability but without the highest reliability, or otherwise) the system should follow the policy described by an administrator". In order to avoid the re-selection of a problematic service the system will manage a black list of them and will use the list to subtract problematic services from the list given by the Registry. This capability should be present in the provider software agents of the Publishing Service.

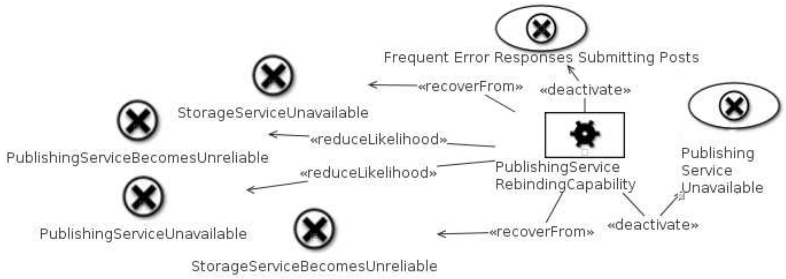

Fig. 5. The Publishing Service Re-binding Capability treating problems diagram

The capability will try to *recover* the system from the *Storage Service Unavailable* and *Storage Service Becomes Unreliable* failures. It also *deactivates* the failure cases caused indirectly by the failures. These "intentions" are related to the self-healing aspect of the capability (see figure 5).

Since the capability selects the service which offers the highest quality level in availability and reliability attributes, it reduces the likelihood of the *Publishing Service Unavailable* and *Publishing Service Becomes Unreliable* failures (see figure 5). Therefore, the capability contributes to the satisfaction of both quality requirements: the high availability and the high reliability of the service (see figure 6).

The capability requires the development of some use cases in order to operate properly (see figure 6). The alternative storage services are found in a Registry, thus the capability needs the location of this registry as input. The *Configure Registry Location* use case describes the functionality of configuring the location of the registry. The capability will manage a black list of services, but problematic services could be healed and the administrator could need remove it from the black list at run-time. But even, the administrator could need add problematic services to the black list because it was detected by another system. The *Manage*

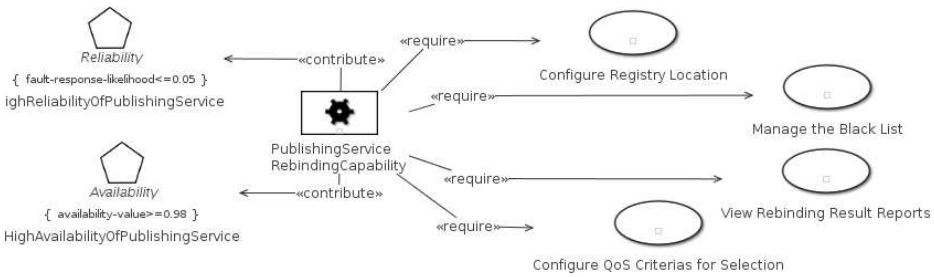

Fig. 6. The Publishing Service Re-binding Capability connected to functional and quality requirements diagram

Black List use case describes this functionality. Also the capability requires the *Configure QoS Criteria for Selection* use case to let the administrator configure what QoS criteria the capability should take into account to select alternatives; and also requires the *View Re-binding Result Report* use case which describes the functionality of viewing, by the administrator, the re-binding logs that are produced at run-time.

This capability is detailed by a self-management process model (figure 7, 8 and 9). There are two monitoring activities: *Storage Service Faults Monitoring* and *Storage Service Availability Monitoring*. The former monitors the number of faults in a period of time in order to update the current value of the quality metric "fault-response-likelihood" of the Storage Service; and the latter monitors the current operational status (available, unavailable) of the Storage Service too. These monitoring activities generate reports which are consumed by the *Analysis Of the Needs to Rebind*. The *Storage Service Faults Monitoring* activity will monitor the using of the Storage Service in order to catch faults, and the *Storage Service Availability Monitoring* could monitor the using, but also could do the monitoring either directly asking to the Storage Service, or subscribing itself to some heartbeat signal.

The *Analysis Of the Needs to Rebind* activity decides if the rebinding is needed or not. The decision is made using the monitoring reports and the quality requirements of the storage service. If the "fault-response-likelihood" metric of the Storage Service is equal or greater than 0.05 the reliability of the Publishing Service will decrease, and if the Storage Service becomes unavailable the Publishing Service will get into failure. Therefore, in these cases the rebinding will be needed. However, the Storage Service could have a notification system that inform to its clients of temporary unavailability for administration purpose. Then the analysis activity could take into account such informations to make a better decision calculating how the estimated period of time in unavailability can affect the quality level of the Publishing Service. Another case where the rebinding is not needed is when there is already a rebinding in execution.

If the rebinding is needed, then the process get into a plan selection phase (see figure 7). The plan selection is made checking if there are available services

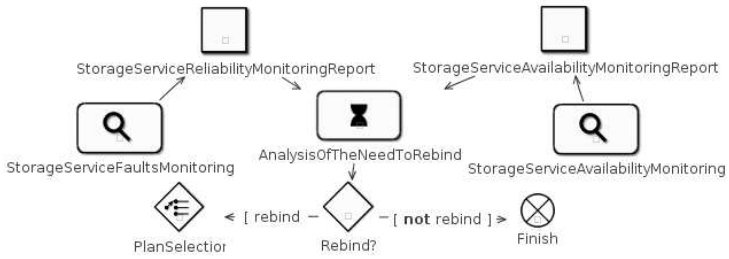

Fig. 7. The Publishing Service Re-binding Process (1) diagram

candidates or not. There are two plans: update the black list, block the using of the publishing service, and do nothing, because there are no candidates; or update the black list and rebind, because there is at less one candidate.

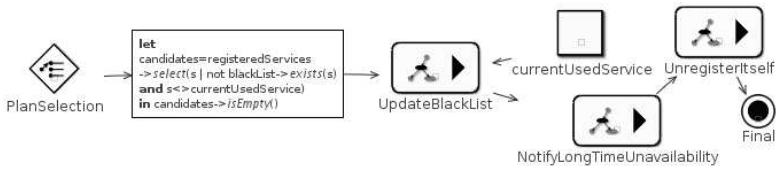

Fig. 8. The Publishing Service Re-binding Process (2) diagram

The figure 8 shows the diagram of the part corresponding to the selection and execution of the plan when there are no candidates. The first elements in the flow is the plan selection guard node that contains the OCL specification which constrains the copy of the token to the outgoing edge. In this case, the constraint is: "if the candidates list is empty then lets the token continue". The rest of elements in the flow are the updating of the black list with the current storage service, the unavailability notification and the unregistering to isolate itself from the rest of the system. The Final Node indicates that the self-management process will stop completely, even the monitoring activities.

The figure 9 shows the diagram of the part corresponding to the selection and execution of the plan when there is at less one candidate. As before, the first elements in the flow is the plan selection guard that constrains the copy of the token to the outgoing edge. When there is at less one candidate in the list the token is copy and the flow continue through the rest of plan step activities. The plan first updates the black list and notifies a temporary unavailability for administration purpose. After that, selects one service according to the given selection criteria and rebinds the publishing service to it. Finally, notifies the availability. The Flow Final Node indicates that the flow stop, but not the process.

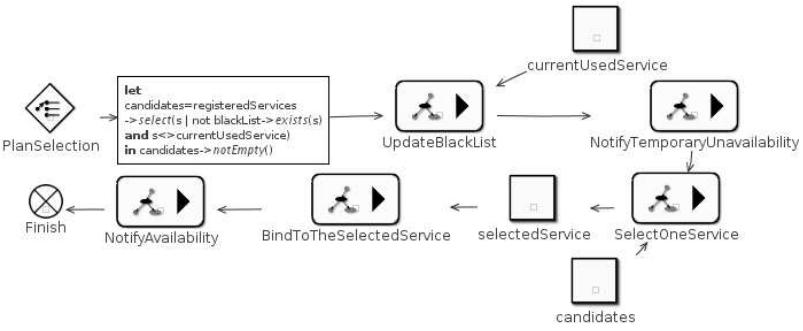

Fig. 9. The Publishing Service Re-binding Process (3) diagram

4 Related Work

The Related Work focus on the works which try to develop a language that can be used to model self-management aspects. There are some works that are more related to the topic of this paper.

One of them is the "UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification" [8]. This one has three parts: an UML Profile and a Catalog for Quality of Service; an UML Profile for Risk Assessment; an UML Profile for Fault Tolerance Mitigation.

The QoS part is for modelling QoS requirements. This profile is limited to quality of service modelling. SelfMML lets developers model not only quality of service requirements but rather any kind of software quality requirements that could be related to self-management.

The Risk Assessment part is to be used for risk assessment modelling, but also included treatment of risks. This part is based on the CORAS method for security analysis[9]. The interesting part for this work is the meta-model and the modelling language of the treatment of risk by use cases and actors. The threats are personified and modelled as actors, the threat scenarios as use cases, and the treatments as use cases too. The connection with QoS is the modelling of risks that could affect the QoS Level of the system. The treatment of risk is specified by use cases. SelfMML makes a difference between a capability of the system to do some management operation by itself and the use cases that such a capability could require. This difference allows the modelling of self-management capabilities that could not require any use case. But also there are cases when a treatment of problems is not well described by an use case but rather by an dedicated entity, e.g. the case study presented in this work, the rebinding could not be well described if only use cases are used.

Finally, the Fault Tolerance Mitigation meta-model and Profile part is for modelling fault tolerance mechanisms for the system. It is mainly for modelling structures that will enable a system to support faults. It includes the modelling of redundancy configuration, monitoring collaborations, fault detection policies, among others. Our work will include fault tolerance, since the fault tolerance

could be considered as part of autonomic computing [3], specially when it is related with self-protection and self-healing. The structure of systems for the maintenance of health is not addressed by the language presented here, but will be considered in future works.

Another work comes from Ian Alexander [7]. This work does not propose a meta-model, what makes that work more informal, but defines a graphical language that suggest a concepts related to some aspect of self-management, e.g. "misuse case", "mitigates", "threat", or "threatens". It was used to inspire part of the work presented here.

Finally, there are several works which present graphical languages for supporting goal-oriented requirement engineering. Some of them are *i**[10], TROPOS [11] and GRL[12]. In these works requirements are identified as goals which can represent functional requirements and non-functional requirements (named as "soft-goal"). Also the languages provide another concepts, which can help in requirement engineering works, such as: actor, task (or plan) and several relationships . Using these languages, self-management requirements can be modelled as goals. However, these languages lack certain elements which can help in the capturing, tracing and specification of these kind of requirements. Some of them are provided by SelfMML, specifically, elements and relationships for modelling failures, their causes and their effects, the *require* relationship, and the elements related to the specification of self-management processes.

5 Conclusion

This work has presented the Self-Management Modelling Language as a language that can be used in the modelling of self-management capabilities in service-oriented systems. Also a case study has been presented to study the application of the language to model self-management requirements in service-oriented systems.

This language has enabled the modelling of a self-management capability in the case study as a requirement in the system, facilitating the capturing and specification of it. However, there are other issues, that could be identified in the case study, which would be interesting to model in service-oriented systems, but the language at this moment did not provide any specific way to do it. The requirement required the specification of some policies in order to make a selection. A model of what policy options the administrator has and how the policies are used to make the selection could be interesting to have. Also, the location of the capability could be inferred by the name of the capability, but it would be interesting to associate the capability to a specific service in a service architecture model. The integration of SelfMML with a language that can be used to model service architectures like Soa Modelling Language (SoaML), could fill this gap. We would like to study both issues as future works in order to develop a more completed language.

A method for realising and verifying self-management requirement specified with SelfMML is an open issue for future work. We would also like to explore model to model transformation from self-management elements to design ele-

ments to support more completely the engineering of self-management capabilities of a target system. Specifically, since agents are suitable to realise self-management capabilities [1,13], self-management elements seem suitable to be mapped to design elements related to agent approaches.

Acknowledgements

This work has been developed with support of the program "Grupos UCM-Comunidad de Madrid" with grant CCG07-UCM/TIC-2765, and the project TIN2005-08501-C03-01, funded by the Spanish Council for Science and Technology.

References

1. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer **36**(1) (2003) 41–50
2. Nami, M.R., Sharifi, M.: Autonomic Computing: A New Approach. In: AMS '07. (March 2007) 352–357
3. Sterritt, R., Bustard, D.: Autonomic computing – a means of achieving dependability? IEEE ECBS **0** (2003) 247
4. : Failure modes and effects analysis. Technical Report MIL-P-1629, U.S. Army (1949)
5. Sindre, G., Opdahl, A.: Eliciting security requirements by misuse cases. In: TOOLS-Pacific 2000. (2000) 120–131
6. Andreas, G.S., Opdahl, A.L.: Templates for misuse case description. In: REFSQ'01. (2001) 4–5
7. Alexander, I.: Misuse cases: Use cases with hostile intent. IEEE Software **20** (2003) 58–66
8. : UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification. OMG. v1.1 edn. (April 2008)
9. Braber, F., Hogganvik, I., Lund, M.S., Stølen, K., Vraalsen, F.: Model-based security analysis in seven steps — a guided tour to the coras method. BT Technology Journal **25**(1) (2007) 101–117
10. Yu, E.S.K.: Modelling strategic relationships for process reengineering. PhD thesis, Toronto, Ont., Canada, Canada (1996)
11. Giunchiglia, F., Mylopoulos, J., Perini, A.: The tropos software development methodology: Processes, models and diagrams. In Giunchiglia, F., Odell, J., Weiß, G., eds.: AOSE'02. Volume 2585 of Lecture Notes in Computer Science., Springer (2002) 162–173
12. Amyot, D., Mussbacher, G.: URN: Towards a new standard for the visual description of requirements. In: Telecommunications and beyond: The BroaderApplicability of SDL and MSC: Third International Workshop, SAM 2002, Aberystwyth, UK, June 24-26, 2002. Revised Papers. Volume 2599 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2003) 21–37
13. Kota, R., Gibbins, N., Jennings, N.R.: Decentralised structural adaptation in agent organisations. In: Organized Adaption in Multi-Agent Systems. Volume 5368 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2009) 54–71

On the Design of a Domain Specific Language for Enterprise Application Integration Solutions *

Rafael Z. Frantz[1], Carlos Molina-Jimenez[2], Rafael Corchuelo[3]

[1] UNIJUÍ University, Department of Technology, Ijuí, Brazil
rzfrantz@unijui.edu.br
[2] School of Computing Science, University of Newcastle, UK
carlos.molina@ncl.ac.uk
[3] Universidad de Sevilla, ETSI Informática - Avda. Reina Mercedes, s/n. Sevilla 41012 Spain
corchu@us.es

Abstract. Enterprise application integrations involve the participation of several existing applications with which the integration solution exchanges data over LANs and the Internet. In these scenarios, operations might occasionally produce exceptional results at runtime due to impairments introduced by the electronic infrastructure such as node crashes, messages lost, delayed or incorrectly composed by applications. To address the problem, the paper suggests a domain specific language to specify the integration solution: it produces platform-independent models and has built-in primitives to produce events that notify of potential exceptional situations. The paper also shows how these events can be processed by an event condition action–based monitor to trigger recovery actions.

Key words: Enterprise Application Integration, Domain Specific Language.

1 Introduction

The computer infrastructure of a typical today's enterprise can be conceived as an heterogenous set of applications (termed the software ecosystem) that includes tens of applications purchased from different providers or built at home. An application is a piece of software that performs an independent and specific business function. Examples of typical functions are calculation of salaries, tax liability, etc. A recurrent challenge that appears in enterprises is the need to enhance the functionality of their software ecosystem by making some of the existing applications to interoperate with others. In the literature, this problem is known as Enterprise Application Integration (EAI) and is all about making two or more existing applications, that belong to the same

* The first author conducted part of this work at the University of Newcastle, UK as visiting member of staff. His work is partially funded by the Evangelischer Entwicklungsdienst e.V. (EED). The second author is partially funded by UK EPSRC Platform Grant No. EP/D037743/1. The third and first authors are partially funded by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, and TIN2008-04718-E).

enterprise, to synchronize their data or to create new functionalities on top of them; in either case, the software that implements the integration is called the EAI solution.

The integration of two or more applications inevitably involves access to each other's data. In the simplest case, an application integration would involve the transmission of a single unit of data (for example, via a RPC) directly from an application to another. However, in practice, application integration normally involves a large number of interactions among applications that, in general, result in a rather complex flow. In these scenarios, the data normally endures some processing between the source and the destination and is subjected to several constraints such as order, timing and validation. For example, data from two or more applications can be validated against syntax errors, then merged and re-formatted to compose a single message, and then delivered to a third application before the expiration of a deadline. To handle this complexity, it is convenient to regard the EAI solution, as a business process that interacts with the participating applications (in this article we call them assets).

From these observations, it follows that the final job of the designer is to produce the EAI solution that implements the new business process on top of the existing computer infrastructure. Two factors make this problem a challenging task: First, computer technology is not static but constantly changing. For example, a piece of software or hardware can be upgraded. A good solution should be able to cope with computer infrastructure evolution, without drastic re-designs. Second, computer infrastructure is far from being 100% reliable. For example, a communication network can fail and delay or lose messages. Again, a good solution should be able to provide continued service – possibly of a degraded quality – despite the occurrence of failures of the infrastructure.

With the above arguments in mind one can argue that a key tool to address the problem of EAI is a specification language and a programmatically way that address the two issues mentioned above.

The Model Driven Architecture (MDA) is a promising software engineering approach suggested by the Object Management Group (OMG) [14]. Its aim is the automation of the software cycle which normally includes design, implementation, deployment, integration, re-design, re-implementation, etc. It relies on the use of tools (as opposite to the conventional manual approach) along the different stages of the software cycle. Central to the MDA are the concepts of model, metamodel and abstraction levels. In brief, a model is the specification of the system under construction, at a given level of abstraction; whereas the metamodel can be a Domain Specific Language (DSL) used to specify the system. The main abstraction levels, in a descent way, are: computation independent (CIM), platform-independent (PIM), platform-specific (PSM) and deployment level. All them provide different models to describe the solution.

A good alternative to address the first challenge presented above is a DSL with built-in constructs to capture the most fundamental concepts involved in EAI solutions such as messages, communication pipes and processing filters, but at a high-level of abstraction. For instance, such languages can be used to describe solutions by means of PIMs, that is, specifications that are implementation neutral and can be programmatically transformed into executable code. Equally important, to address the second issue, such a language should offer a means for capturing exceptional situations likely to have an impact on the EAI solution. To the best of our knowledge and in accordance with

results from previous research [6], DSLs with these highly desirable features are still a research topic. We are aware that there are some preliminary results in this direction. For example, in [6] the authors present Guaraná – a DSL designed to produce PIMs; in other words, it addresses the first issue hinted above; a limitation of Guaraná is that as it is, it can only specify normal execution flows; in other words, it does not address the second issue since it does not have any mechanisms to specify exceptional situations. As an alternative to cover the gap, we suggest the enhancement of Guaraná with constructs for detection and handling of exceptional situations. In pursuit of this goal, we show in this paper how Guaraná ports can be enhanced to signal exceptions when communication operations (e.g. read, write, solicit) executed by ports against an asset fail to complete, or a validation test on received data produces abnormal results. We target port operations first because we consider that they are the most fault-prone operations in an EAI solution. Also, we show how a conventional Event Condition Action (ECA) mechanisms can be used to handle these exceptions. Several engineering requirements combine to make EAI a hard problem; before tackling them, it is worth clarifying what requirements we take on board in our research and what assumptions we make.

The first requirement we account for is that the existing assets are and should continue to be functionally independent from each other with and without the EAI solution in operation. Mutually-dependent assets fall outside the scope of this paper. It follows that our EAI solution should provide only exogenous coordination. To meet this requirement we rely on loosely-coupled interactions between the solution and the assets.

Another typical requirement on the EAI solution is that it should not involve the modification of the code or configuration of the original asset; the implication of this restriction is that the EAI solution can count only on the original interfaces that the participating assets offer, to make them interoperate.

We make no assumptions about the physical locations of the assets. They can be located within the same building and be linked by a LAN or in different continents and communicate over the Internet. The involvement of the Internet presents the designer with additional challenges: Internet communication is far from being fully reliable; it can lose and duplicate messages; and introduce unpredictable delays. Without due attention, these impairments can render an EAI solution unoperational. Communication delays become highly relevant in EAI solutions with strict and tight time constraints.

Since we focus on EAI, we can leave authentication and security issues out of the equation as these problems are not of major concern in these scenarios. Likewise, we can assume that access to data between assets is always granted, likely under some restrictions of no relevance to our discussion.

This paper is structured as follows: Section 2 places our research in context and discusses the related work; Section 3, offers a brief introduction to our DSL Guaraná and places it within the context of the model driven approach; Section 4, is the heart of our paper – it discusses our failure semantics and shows how Guaraná's ports can handle exceptional situations; Section 5, presents a realistic scenario of enterprise integration that we use as a validating example, and discusses our event condition action-based exception handling mechanism to support Guaraná. Finally, we draw conclusions and future work in Section 6.

2 Related Work

The UML-profile [13] for EAI solutions is directly related to our work on Guaraná and is arguably an attractive alternative. We rule it out on the basis that UML profiles are basically extensions to UML intended to cover the limitations of the native UML; unfortunately, the UML-profile for EAI solutions has not been very successful due to its complexity and lack of expressiveness (see [1] for a discussion on the adequateness of using UML-profiles to represent DSLs).

The Business Process Modeling Notation (BPMN) [18] can be used for specifying EAI solutions but at CIM level, that is, at a level that is too abstract for EAI designers. Related to our work are also EAI technologies like BizTalk [10], Mule [12] and Camel [5] which produce technology-dependent solutions: in the context of MDA, BizTalk fits the platform-specific level while Mule and Camel belong to deployment level. We regard BPMN, BizTalk, Mule and Camel as complimentary to Guaraná, rather than competitive.

Our work is closely related to the on-going research on exception handling in Web services composition. Of interest to us is the discussion presented in [19]. We share with the authors the idea of using ECA policies to handle exceptional events. However, the paper studies the problem at BPEL level of abstraction and suggests the use of an integrated exception handling mechanism with the intention of conducting execution planning to prevent the occurrence of exceptions. Our goal is different – we address exceptions at PIM level of abstraction and propose mechanisms to recover from exceptional situations rather than to prevent their occurrences. A similar discussion on exception handling at BPEL level and complimentary to [19] can be found in [9]. In [4] the authors propose a policy-driven middleware solution (implemented in .NET and manually portable to other platforms) to handle exceptions in web service composition; we consider this a valid result but too implementation specific, since our interest is in abstract PIMs. With this paper we share the view that communication operations are the most fault-prone. Relevant to us is also the classification of faults which can roughly be mapped into the exception that Guaraná's ports can detect.

An illuminating discussion about the complexity of handling exceptional situations in EAI, such as the unexpected cancellation of an operation due to infrastructure failures or human related events, is presented in [7]. Authors argue that to be effective, a compensation mechanism should take into consideration the state of the two interacting applications. As the discussion is at conceptual level, the authors present no solution.

3 An Overview of Guaraná – a Domain Specific Language for EAI

A DSL is a well focused language developed to address problems in a particular domain. It provides a set of dedicated abstractions, elements and notations with formalization to assist the designer in expressing its solution at the level of abstraction of its DSL. In MDA, a given model is programmatically transformed into a model of a lower level of abstraction. Models of high-level of abstraction are implementation neutral and called PIMs; whereas models of low-level of abstraction are implementation specific and called PSM. There are estimations [17] that show that for each dollar spent

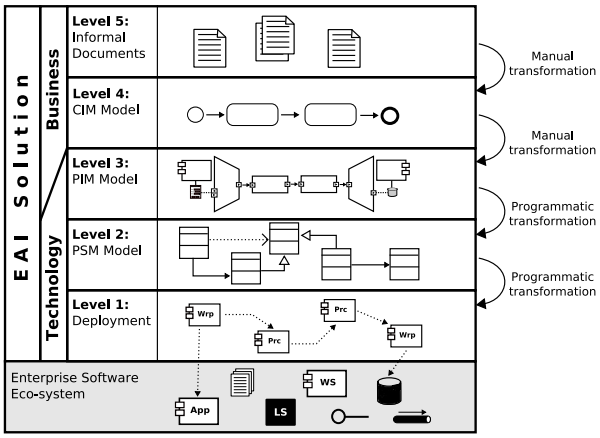

Fig. 1. Abstraction levels for an Integration Solution.

on developing an application, companies usually spend from 5 to 20 dollars to integrate it into EAI solutions. From this perspective, the MDA approach looks like a promising alternative to cut these costs down. For instance, the designer can produce a PIM model of the EAI solution that can be re-used to automatically derive as many PSMs as necessary to match technology evolution within the enterprise.

The feasibility of the MDA approach depends on the availability of the source metamodel, target metamodel, transformations, etc. For instance, in the EAI domain a DSL is still a miss. We suggest Guaraná as an alternative to cover the gap. Guaraná produces graphical designs of EAI solutions at a high-level of abstraction; it uses Enterprise Integration Patterns (EIP) [8] and covers currently existing gaps in the field.

3.1 Abstraction Levels for Integration Solutions

In the MDA, the specification of an EAI solution can undergo several transformations through different levels of abstractions before it is finally implemented on top of the software ecosystem. This idea is shown in Fig. 1 with the intention of placing Guaraná within this context. The figure shows five levels of abstraction separated into business and technology. Level 5 is the most abstract one and is entirely technology-specific solution neutral in contrast, level 1 is the final technology-specific deployable solution. Guaraná belongs to the third level.

LEVEL 5: Models are produced by business analysts and provide an informal description of the problem at a high-level of abstraction, that is, with no notion of applications, message flows or technology like software ecosystems. Models are specified in natural language and normally suffer from imprecisions, omissions and ambiguities.

LEVEL 4: Here models are considered CIMs. They are refinements of models from level 5, produced manually by business analysts and expressed in standards like BPMN [18]. The notions of participating applications (source and consumer of data) and message flow appear at this level; yet the model does not capture core domain

specific design concepts like the internal structure of the solution or the communication with its applications. The absence of these concepts prevents their programmatic transformation into executable models.

LEVEL 3: Models of this level are produced from models of level 4 manually by system analysts with expertise in EAI solutions. They precisely specify the functionality and structure of the solution in a manner that it can be programmatically converted into PSMs of level 2. So they deal with concepts at the granularity of applications, processes, tasks, message flow, integration links, ports, wrappers, etc. These models are expressed in a DSL with built–in constructs to describe all the EAI specific concepts listed above.

LEVEL 2: Models of this level result from automatic conversion of models from level 3; they are PSM and can be mapped into executable code of the chosen technology.

LEVEL 1: Models of this levels are the actual executable code of the solution and are programmatically generated from models from level 2.

3.2 Guaraná Constructors

Guaraná provides a set of domain specific constructors to design EAI solutions. This language not just introduces and describes this domain specific concepts, but also provides a very expressive and needful graphical notation for this constructors, that allow to visually design an EAI solution. Below we introduce the main constructors and in Fig. 3 we provide an example of a designed EAI solution with Guaraná where the use of these constructors can be seen. **Decorators** are used to provide visual information about the participating assets and their layer(s). They do not have an influence on the executable model. **Building blocks** represent the processing constructors of the EAI solution and are composed of tasks. There are two types of building blocks in Guaraná: wrappers and processes. A **wrapper** communicates the EAI solution to an asset, so it contains communication-specific tasks and has a port connected to a decorator. **Processes** model the essential services of the EAI solution, so they contain integration-specific tasks; they are connected (through ports and integration links) to each other and/or to wrappers. **Slots** are memory buffers used within building blocks for port to task and task to task internal communications. **Tasks** are message processing constructors and appear inside processes and wrappers. A task reads messages from incoming slots, processes them (e.g. enriches, translates, filters, etc.) and deposits the result in the outcome slot. **Ports** are used to communicate the internal building blocks of an EAI solution and the EAI solution with its assets. **Integration links** are channels that transport messages between building blocks. They are used to connect the entry/exit ports used by building blocks.

4 Failures in Application Integration Solutions

The general assumption we make about the reliability of the components involved in an enterprise application solution is that they will occasionally fail. Thus our goal is to provide the enterprise application integration with mechanisms to tolerate the occurrence of failures as opposite to prevent their occurrence. For this to be possible, we need to identify the failure behavior that the enterprise application integration are likely

to exhibit. The failure behavior is also known as failure model or failure semantics [2] and stipulates what kind of errors the system (the enterprise application integration for instance) will be able to tolerate: detect at runtime, execute corresponding recovery action and return to the normal execution flow possibly with a degraded performance. An application EAI solution can fail in different ways, non-surprisingly, different authors suggest different classes of failures (see for example [4], [9]); yet it seems to be a general consensus that most of the failures that impact EAI solutions emerge from the execution of operations that involve exchange of data with assets; the reason being that these operations normally involve network communication and possibly over unpredictable channels like the Internet. The EAI solution can run distributed in different machines, so in this case network problems also should be considered for building blocks' communication. This type of errors are not addressed in this paper. On this basis and to comply with space restrictions, this paper focuses only on two types of failures that might occur inside ports used by the EAI solution to communicate with its assets: omission failures and response failures. Yet it is worth mentioning that our idea is general enough to be expanded to other operations executed by the EAI solution.

Omission Failures (OMF): In our communication model we assume that once a communication operation (read, write and solicit-response) is started by the EAI solution, it terminates within a strictly defined time interval and declared by the EAI solution either success or failure. The failure result models situations in which the network and asset problems might prevent the solution to send or receive a piece of data to/from the asset within the deadline interval, when this happens we say that the asset has exhibited an omission failure. Notice that in our communication model operations completed beyond the time constraint are taken as failure, so data received by a read operation after the expiry of the deadline is ignored. In our discussion we use *OK* and *OMF:NOK* to represent success and failure, respectively.

Response Failures (REF): As suggested by leading standards in e-business middleware like RosettaNet [15] and/or by well known integration technologies like BizTalk [11], it is not enough to receive a response in time as the responder, the asset in this case, might respond incorrectly. Thus a received message has to satisfy some syntactic validation tests (e.g., headers and body inspected and understood) before it can be taken by the EAI solution for processing. This kind of failures are known as response failures. To model them we run a validation test on every message received by the EAI solution, that produces either success or failure. Again, we use *OK* and *REF:NOK* to represent success and failure, respectively.

4.1 Exceptions of Guaraná Ports Operations

Guaraná provides four types of ports for communication: one-way EntryPort, one-way ExitPort, two-way SolicitorPort and two-way ResponderPort. One-way EntryPorts are used for reading messages in an internal EAI solution communication and from assets, as well; one-way ExitPorts are used similarly, but for writing. Two-way SolicitorPorts are used to solicit data from assets in solicit-response mode; in principle this operation can be split into two individual operations or abstracted as single atomic one; for simplicity we discuss only the latter case. Two-way ResponderPorts are irrelevant in our arguments and not discussed further.

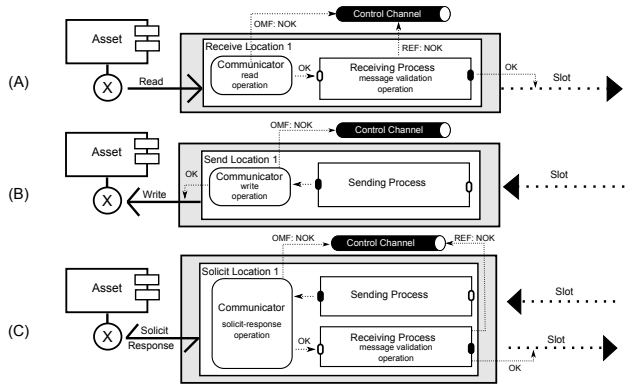

Fig. 2. (A) Entry Port, (B) Exit Port, (C) Solicitor Port.

The instrumentation of our failure model in Guaraná's ports is shown graphically in Fig.2. EntryPort and SolicitorPorts can contain one or more locations with their respective communicators. Each location is associated to a single source of data (e.g., application layer such as database, file, user interface). We assume the existence of a single location as this is enough to explain our ideas. The actual communication operation (read, write and solicit-response) is performed by the communicator; so to implement our omission failure model, we provide communicators with the notion of deadline to complete their operations. Read messages are delivered by communicators to the receiving processes. The sending processes are irrelevant in our discussion yet we can mention that they process the message destined to assets, before passing it on to the communicators. To model response failures, we include a validation operation inside receiving processes. As show in the figure, in response to a given operation (e.g., take read operation of EntryPort A) communicators produce either OK or OMF:NOK. The OK message represents the normal response and is fed into the normal flow, whereas OMF:NOK represent the abnormal result and is notified to the control channel. Similarly, the validation operations produce either OK (fed into the normal flow) or REF:NOK which is notified into the control channel.

5 Validation

To validate our ideas we will show how both the normal and exceptional execution flow can be specified in Guaraná.

5.1 Example

To set the scene, we will use the scenario of an application integration problem under study at Unijuí, Brazil. Apart from some small modification introduced to highlight the issues of our research interest, the scenario is realistic. The project involves five assets: a Call Center System (CCS), Payroll System (PS), Human Resources System

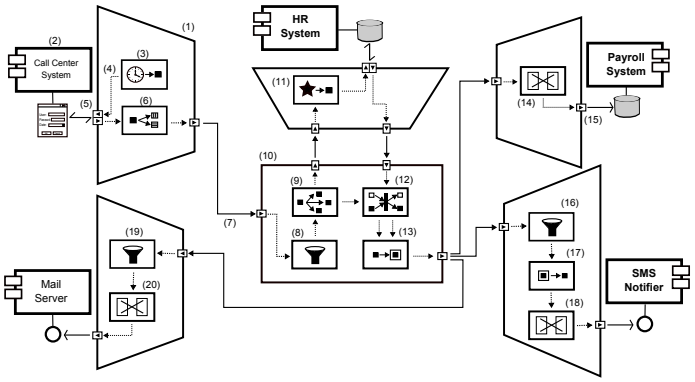

Fig. 3. UNIJUÍ's integration solution with fault-tolerance.

(HRS), E-mail Server (ES) and Short Message Service (SMS). The CCS manages the university telephone system. The PS processes the monthly payments of the employees. The HRS manages the personal data of the employees. The ES runs the e-mail service. SMS runs a service for sending text messages to mobiles phones. The problem can be informally described as follows:

1. Employees use the phone facilities of the university for making external phone calls of both business and personal purposes. All calls are recorded by CCS.
2. Business calls are free. The cost of personal calls is deducted from the caller's salary which is processed by the PS on the last day of the month at 9:00 a.m.
   – *No calls can be deducted before notifying the caller by e-mail, SMS text or both.*
3. To deduce the cost of a call, the PS requires: 1) the caller's name, personnel number, e-mail and mobile phone number, from the HRS; and 2) the cost and destination of the call, from the CCS.
4. To guarantee that a call will be deducted from the current month's salary the PS needs to receive the input by 8:00 a.m. on payment day.
   – *Input received after this time is logged and processed in the next month.*

The description has two salient features: 1) It includes operations with strict time constraints, for example, "input by 8:00 a.m. on payment day". 2) It accounts for potential exceptional situations and separates the normal execution flow (normal text) from the exceptional one (italic text). This problem can be solved by using the exception mechanism introduced in Section 4.

5.2 Integration Solution

The Guaraná specification of our example is shown in Fig. 3; ignore for the time being, deadline constraints and potential exceptions. The integration flow is started by the timer task (3) located inside the wrapper (1) that communicates with the CCS – represented by a decorator (2). This task creates an activation message every t units of time (e.g. five minutes) and writes it to a slot (4). The message activates a solicitor

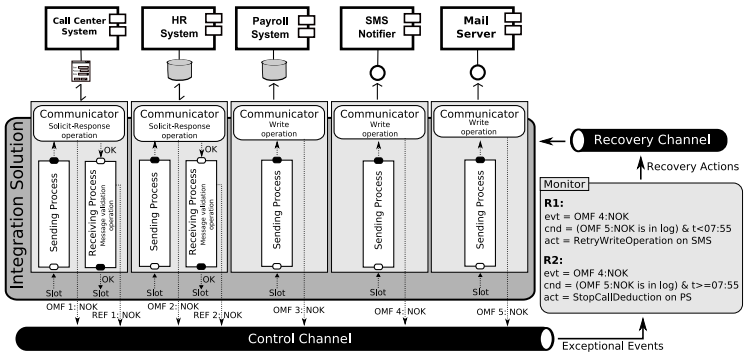

Fig. 4. ECA - based exception handling mechanisms.

port (5) that extracts all phone calls made in the last five minutes. The CCS offers only a user interface, so the solicitor port uses a scrapper (a technique for extracting information through a user interface) to interact with the CCS. The splitter task (6) breaks the message into individual messages containing just one phone call each. These messages are sent, through an integration link (7), to the central process (10). Messages related to irrelevant (e.g., free or insignificant cost) calls are filtered out by a filter (8). Messages of interest are replicated (9) to the wrapper of the HRS and to a correlator (12). The HRS is queried by the message created by a custom task (11); the response from the HRS is sent to the central process (10) where it enriches the original message by means of an enricher (13). The enriched message is sent after the corresponding translations (14, 18 and 20) to match destination formats, to the PS, SMS and ES. Messages that, for some reasons, do not contain a phone number or an e-mail address are filtered out (respectively by filters 16 and 19). A slimmer (7) shortens messages down to the short text standard before sending them to the SMS.

5.3 An ECA-based Exception Handling Mechanism

To handle exception occurrences at programming level and independently from the normal execution of the EAI solution, we suggest the use of an exception handling mechanism based on the conventional Event Condition Action (ECA) paradigm. We show its functionality with the help of Fig. 4 which is a simplified version of Fig. 3 meant to highlight port operations between the EAI solution and the five assets involved.

Our mechanism involves a monitor and a control channel that links the monitor with the EAI solution (say by a publish/subscribe paradigm). Also, the monitor is linked to the EAI solution by the recovery channel for recovery actions. The EAI solution represents the normal execution flow (normal text of our validating example) whereas the monitor represents potential exceptional executions (italic text of the example). The monitor is instrumented with ECA rules that execute exceptional actions upon receiving exceptional events through the control channel. As shown in Fig. 4, exceptional results (OMF:NOK and REF:NOK) from the execution of port operations (solicit-response, message validation and write) are published to the control channel. The normal results

of the operations are fed only into the normal execution flow, but if necessary, they can also be published to the control channel to be used by the monitor.

The exceptional actions are application-specific and fall outside the scope of this paper; yet we can briefly mention that they are recovery actions whose execution brings the control flow back to the normal execution. For instance, an operation can be simply ignored, retried, etc. Some typical recovery patterns are discussed in [9] and [3]. The rules can be written in a rule language (see for example [4]) and executed by a conventional rule engine supported by timers, event-log files, queues, etc., (see for example [16]). The discussion of these details falls outside the scope of this paper. The rules shown inside the monitor are only illustrative and far from being complete; they are meant to show how exceptional situations can be handled; to save space we will focus only on the exceptional execution flow of point 2 of our example: *No calls can be deducted before notifying the caller by e-mail, SMS text or both*.

In our simplified notation R, evt, cnd, act and t stand for rule, event, condition, action and time respectively. The $07:55$ represents the time on the payment day to notify the PS to stop it from processing a call deduction when the caller has not been notified. Similarly *OMF 5:NOK is in log* checks for the existence of OMF 5:NOK records in the log file of the rule engine. $R1$ captures the possibility that the *write* operation against the SMS fails; the condition checks if the notification by e-mail has failed and if there is time to retry the SMS operation; when the exceptional event OMF 4:NOK is received and the condition holds, the recovery action *RetryWriteOperation on SMS* is executed. $R2$ is complimentary to $R1$ as it also reacts to the OMF 4:NOK exceptional event; yet it is triggered when it is time ($t >= 07:55$) to notify the PS of the problem and executes the *StopCallDeduction on PS* recovery action to stop the PS from processing the call under question.

6 Conclusion and future work

It is worth emphasizing that at the current stage of our research the failure semantics of Guaraná's ports consider only omission and response failures that may be raised due to a local time constraint at port level when ports interact with assets. Our future work is to enhance Guaraná's failure semantics also with the capability of capturing message processing failures that may occur into the sending process element at ports (see Fig. 2). This element can be used to validate the message before forwarding it to the communicator so that assets are prevented from receiving and processing invalid messages that will only produce NOK outcomes and exception signals. Another idea is to extend this idea to the whole EAI solution and thus considering those message processing failures that may occur within building blocks, since the principle here is the same as for sending processes; in principle each building block involved in the flow might produce success (OK) or failure (NOK) after processing a message.

Considering the whole EAI solution, the failure semantic could be enriched with the notion of a global deadline (global time constraint). This kind of constraint should specify a time-to-live for messages, meaning the message is valid and can be normally processed by the EAI solution, as well as, be delivered to the target asset(s) within this time. In this case a corresponding class of failure could be raised and handled, if the EAI

solution did not meet the global time constraint. There would be a direct relationship between these two kinds of constraints, since the total amount of time in local time constraint could give a hint to build the global time constraint.

The paper recognizes the need to account for exceptional situations that normally impact EAI solutions at run time and suggests an approach to capture them at an abstract level of the specification. To address the problem and in support of the model driven approach to cope with computer technology evolution, the paper contributes with an innovative DSL that 1) produces PIMs; and 2) whose operation (ports at current stage) invocations account for exceptional outcomes: they either produce a normal result or an exceptional event that is processed by an event condition action-based monitor that triggers recovery procedures.

References

1. A. Abouzahra, J. Bézivin, M.D. Del Fabro, and F. Jouault. A practical approach to bridging domain specific languages with UML profiles. In *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA*, volume 5, 2005.
2. F. Cristian. Understanding fault–tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.
3. V. Ermagan, I. Kruger, and M. Menarini. A fault tolerance approach for enterprise applications. *Services Computing, 2008. SCC '08. IEEE Int'l Conf on*, 2:63–72, July 2008.
4. A. Erradi, P. Maheshwari, and V. Tosic. Recovery policies for enhancing web services reliability. In *Proc. Int'l Conf. Web Serv.*, pages 189–196, 2006.
5. Apache Foundation. Camel Home, 2008.
6. R.Z. Frantz, R. Corchuelo, and J. González. Advances in a DSL for Application Integration. In *Proceedings of the Zoco'08 Workshop*, pages 54–66, Gijón (España), 2008.
7. P. Greenfield, A. Fekete, J. Jang, and D. Kuo. Compensation is not enough. In *EDOC '03: Proceedings of the 7th International Conference on Enterprise Distributed Object Computing*, page 232, Washington, DC, USA, 2003. IEEE Computer Society.
8. G. Hohpe and B. Woolf. *Enterprise Integration Patterns - Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2003.
9. A. Liu, Q. Li, L. Huang, and M. Xiao. A declarative approach to enhancing the reliability of bpel processes. In *Proc. IEEE Int'l Conf. Web Services*, pages 272–279, 2007.
10. Microsoft. Microsoft BizTalk Server 2006 R2 Home, 2008.
11. Microsoft. BizTalk Framework 2.0: Document and Message Specification, Dec 2000.
12. Inc. MuleSource. Mule 2.x User Guide, 2008.
13. Object Management Group (OMG). OMG EAI Profile Home, 2004.
14. Object Management Group (OMG). OMG Home, 2009.
15. Rosettanet. Rosettanet: Implementation framework–core specification, version: V02.00.01, revised 6 mar, 2002.
16. M. Strano, C. Molina-Jimenez, and S. Shrivastava. A rule-based notation to specify executable electronic contracts, cs–tr no. 1115. Technical report, School of Computing Science, Newcastle University, 2008.
17. J. Weiss. Aligning relationships: Optimizing the value of strategic outsourcing. Technical report, IBM, 2005.
18. Stephen A. White. Business Process Modeling Notation (BPMN) Specification 1.0, 2009.
19. L. Zeng, H. Lei, J j. Jeng, J-Y. Chung, and B. Benatallah. Policy–driven exception-management for composite web services. In *Proc. Seventh IEEE International Conference on E–Commerce Technology (CEC05), 19–22 July*, pages 355–363. IEEE Computer Society, 2005.

Tool support for Service Oriented development from Business Processes

Andrea Delgado[1], Ignacio García-Rodríguez de Guzmán[2], Francisco Ruiz[2], Mario Piattini[2]

[1] Computer Science Institute, Faculty of Engineering, University of the Republica, Julio Herrera y Reissig 565, 1300, Montevideo, Uruguay
adelgado@fing.edu.uy
[2]Alarcos Research Group, Dep. of Information Technologies & Systems, University of Castilla - La Mancha, Paseo de la Universidad 4, 13071, Ciudad Real, España
{ignacio.grodriguez, francisco.ruizg, mario.piattini}@uclm.es

Abstract. The integration of Business Process Management (BPM), Service Oriented Computing (SOC) and Model Driven Development (MDD) paradigms to improve the development of services oriented solutions from business models is nowadays in the spotlight. Organizations wanting to remain competitive despite the constant changes in their business are paying more attention to their business processes and its base lifecycle. Business process modeling is also at the centre of software development efforts, as making those models explicitly constitutes the basis for services definition. Transformations between business process and services models allow the automatic generation of services from business processes in a repeatable and systematic way, easing the development process. In this paper we present MINERVA's tool support for service oriented development from business processes, including QVT transformations from BPMN to SoaML models to automatically generate service models from business process models.

Keywords: Service Oriented Computing (SOC), Model Driven Development (MDD), Business Process Management (BPM), business processes, service oriented methodologies, model transformations, tool support.

1 Introduction

The development of services oriented solutions to realize business process provides organizations with the needed organizational agility to react to changes, allowing performing changes to each one –business and software – with minimal impact in each other. The integration of Business Process Management (BPM) [1][2], Service Oriented Computing (SOC)[3][4][5] and Model Driven Development (MDD) [6][7][8] paradigms to improve the development of services oriented solutions from business models is nowadays in the spotlight. Organizations wanting to remain competitive despite the constant changes in their business are paying more attention to their business processes and its base lifecycle as defined in [9][10][11]. The implementation of business processes with services also contributes in reducing the

gap between the areas of business analysis and Information Technology (IT), easing communication and understanding of business needs. The model driven development supports the definition and maintenance of the relationship between the various models involved, and automates as much as possible the passage from one another by means of transformations. The main objective of the ongoing research work is to provide support to the continuous improvement of business processes based on their lifecycle, applying SOC and MDD paradigms to business process to enable the needed organizational agility. This vision is expressed in MINERVA [12] (Model drIveN and sErvice oRiented framework for the continuous business processes improVement & relAted tools) which is a framework comprising elements in three dimensions: conceptual [13], methodological [14] and tool support, including the Business Process Maturity Model (BPMM) [15] and measures [10][16] for the design and execution of business processes, to guide the improvement effort.

In this article we present MINERVA's tool support for service oriented development from business processes in three main aspects: a method plug-in developed in Eclipse Process Framework (EPF) [17] Composer to support the defined Business Process Service Oriented Methodology (BPSOM) [14]; Query/Views/ Transformations (QVT) [18] transformations from Business Process Modeling Notation (BPMN) [19] models to Service Oriented Architecture Modeling Language (SoaML) [20] models -based on the defined ontology [13]- to generate services automatically, and the technical architecture and selected tools to support MINERVA's lifecycle. The rest of the article is organized as follows: in section 2 the EPF Composer method plug-in to support the methodological approach is described, in section 3 the technical architecture and tools selection of MINERVA are presented, in section 4 the defined QVT transformations are described along with an example, in section 5 related work is presented and finally in section 6 conclusions and future work are discussed.

2 BPSOM method plug-in

The methodological approach defines a Business Process Service Oriented Methodology (BPSOM) [14] as a plug-in to be incorporated in the software base development process used in the organization. The methodology is defined as a method plug-in and published as a web site [21] using the EPF Composer to provide interoperability with other processes defined in the same way. BPSOM defines activities, roles, work products and its templates in three Disciplines defined as key for service oriented development: Business Modeling, Design and Implementation. In Fig. 1 several elements of the methodology are shown, on the left side the defined categories can be seen: Disciplines, Work products, Roles and Lifecycle, along with some of its comprising elements such as activities, tasks, deliverables, roles. On the right side an example of tasks definition is presented for BM2 - Identify Business Processes, showing participating roles, work products defined as inputs and outputs, purpose, description and Discipline that comprises it. Due to space reasons, a brief description of BPSOM is presented here, the complete definition is presented in [14].

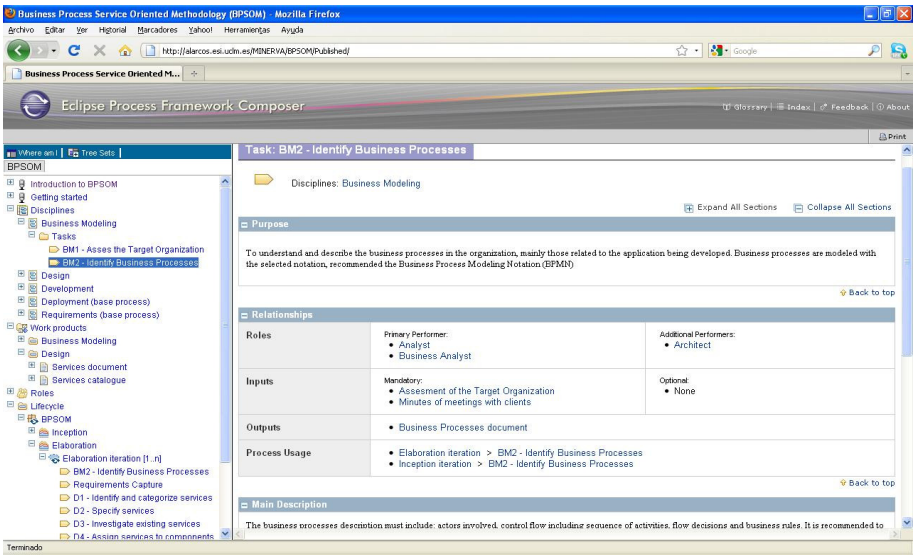

Fig. 1. Global view of BPSOM web site created using EPF composer

The defined lifecycle is iterative and incremental following the unified process[22] incorporating the four phases: Inception, Elaboration, Construction and Transition. The activities are inserted in the phases of the base software development process, making the corresponding adaptations when needed. The defined lifecycle serves as a guide to indicate the emphasis on the realization of activities at each stage of development. As an example, the activity's worfklow for the defined Elaboration iteration is shown in Fig. 2.

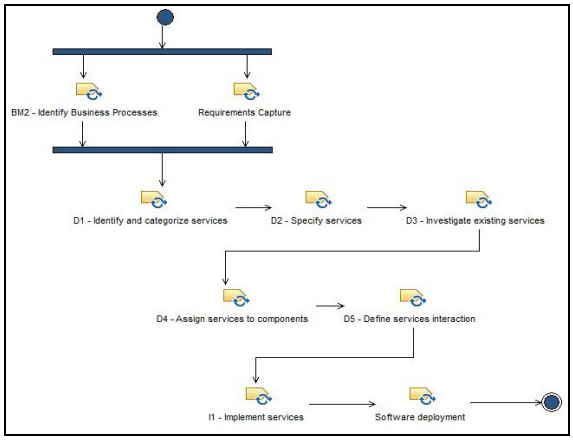

Fig. 2. Activity's workflow for an Elaboration iteration in BPSOM

Disciplines and activities. BPSOM activities are defined in three key Disciplines to guide the service oriented development effort: Business Modeling, Design and Implementation. These activities have to be integrated with other existing activities of the base development process, such as requirements capture, architecture definition, service testing and deployment, and project management.

Business Modeling. The Business Modeling Discipline aims to understand and describe the business processes in the organization, mainly those related to the application being developed. Business processes are modeled with the selected notation, recommended the Business Process Modeling Notation (BPMN). It also promotes the involving of the project team with the organization for which the development is being carried out, in issues such as: the area of business, operation, employees, etc. of the organization. There are two activities defined to reach these goals: BM1 – Asses the target Organization and BM2 – Identify Business Processes.

Design. The Design Discipline adds the following goals to the ones defined for generic software design: to identify and specify the services needed to perform the business processes modeled in the Business Modeling Discipline, classifying them by type of service, to generate and maintain a services catalogue for services reuse in the organization (functionalities, components), and to define the services composition (orchestration, choreography) needed to realize the identified business processes. Five activities are defined to reach these goals: D1 – Identify and categorize services, D2 – Specify services, D3 – Investigate existing services, D4 – Assign services to components and D5 – Define services interaction.

Implementation. The main goal of the Implementation Discipline is to get the identified services implemented and working as defined. To reach this goal the activity I1 – Implement services is defined.

Work products. Work products provide the basis for controlling development progress and reaching the defined goals of the activities, Disciplines and lifecycle iterations. The defined work products in BPSOM are as follows: Assessment of the target organization, Business Processes document, Services document, Services catalogue and Services implemented. Each document has a defined template to guide its realization, defining its content.

Roles. The roles in BPSOM are a selection of base roles that we consider to be the more important ones for service oriented development based on business processes: Analyst, Business Analyst, Architect and Developer. Each role performs several activities as primary role, and is responsible for some work products realization. The Business Analyst is explicitly included now to emphasize the importance of the participation of Business people in the Business Modeling Discipline.

3 Technical architecture and tools

To support service oriented development from business processes following BPSOM methodology, a selection of existing tools is given. The aim of MINERVA framework is to integrate free existing tools facilitating the development using BPSOM; for every activity a tool is recommended for its realization. So far we are working with an integrated selection of tools available in the Eclipse IDE: Eclipse BPMN Modeler [23] for business process modeling, Medini QVT [24] Eclipse plug-

in for business process to services QVT transformations, Magic Draw Cameo SOA+ [25] Eclipse plug-in for SoaML modeling and ModelPro [26] Eclipse plug-in for JEE code generation. The technical architecture and the first selection of tools are shown in Fig. 4, to support the complete MINERVA's lifecycle.

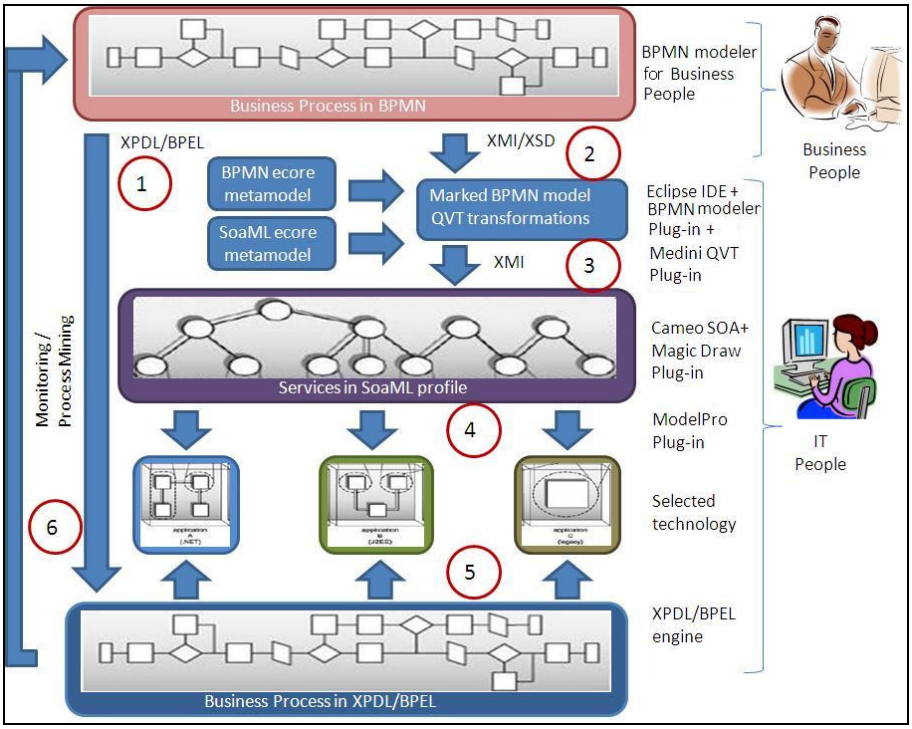

Fig. 4. MINERVA tool support for services development from business processes

As it can be seen in Fig. 4 and according to BPSOM, the first activity shown corresponds to the business process modeling that is done by Business People, in a BPMN modeler. After the business process model is defined, in step (1) the business process execution model is obtained expressed in XPDL [27]/BPEL [28], and in step (2) the business process model is exported in XMI/XSD format, so it can be imported for IT People in the Eclipse IDE. Several tools allow these kinds of export files from business process models, and with the release of BPMN 2.0 [29] we expect the format would be homogeneous for all tools. After the BPMN model is imported in Eclipse IDE, activities that will be transformed to services have to be marked as of "Service" type according to the service design activities from BPSOM, defining its input and output messages, operations, implementation, among others. Running the QVT transformations defined in Medini QVT plug-in in step (3), the corresponding elements in SoaML profile are obtained. The inputs for the QVT transformations are the source model in XMI format that is the XMI file associated with the BPMN business process model, and the two corresponding metamodels: BPMN Modeler and SoaML metamodels in ecore format, from which the QVT transformations are defined. Then the target XMI file corresponding to the SoaML model that is generated by the transformation is loaded into Magic Draw Cameo SOA+ plug-in,

and in step (4) using ModelPro plug-in the code is generated from the obtained SoaML model. Step (5) represents the invocation of the generated service components from the business process execution engine. Finally, in step (6) monitoring of the business process execution and its evaluation by means of techniques such as Process Mining [30] over log files is done, using tools as Prom [31] and its analysis plug-ins.

Business Process Modeling Tools. For business process modeling we are evaluating several tools that implements BPMN standards, including BizAgi modeler which allows XPDL export, Visual Paradigm Business Process Architect which allows XMI export, as it is desirable that Business People could model the business processes to give them to IT People. For our proof of concept we are using directly the Eclipse BPMN modeler as we want to focus on QVT transformations.

Service Oriented Modeling Tools. There are few implementations of the SoaML specification yet that can be seen in [32]. Although we are analyzing the functionalities provided by each of them along with it ease of use, we have integrated the MagicDraw CAMEO SOA+ and the ModelPro Eclipse plug-ins for the proof of concept, as we are using Eclipse as base platform. SoaML implementations provide the needed stereotypes to specify SoaML models, and to generate code in the desired technology. We are planning our own implementation of SoaML as Eclipse plug-in.

Code generation from services in SoaML. The ModelPro plug-in generates JEE code from SoaML models, providing several examples. For the generation of code for our own example we need to add elements to the generated SoaML model, via new defined transformations or manually, to provide the MDA engine with the required information for doing the generation of code.

4 QVT transformations

BPSOM methodology guides the derivation of services from business processes by means of activities, roles and work products definition. To improve the development process, the automatic generation of services oriented models from business process models is added, which is presented in the following.

4.1 QVT transformations definition

To define QVT transformations between service models and business process models, we previously defined an ontology [13] in the conceptual dimension of MINERVA, in which we identified the relations between these models, in order to understand what elements we want to obtain in service models from elements in business process models. Following the Model Driven Architecture (MDA) [8] standard, we transform BPMN models into SoaML models by defining QVT transformations from a BPMN metamodel as origin metamodel, to the SoaML metamodel as the target metamodel. Then, a BPMN model compliant with the BPMN metamodel will be transformed into a SoaML model compliant with the SoaML metamodel.

There are many elements in each metamodel to be related, so to begin with we selected a sub-set of key concepts and relationships to define the QVT transformations that allow us to obtain the structural view of services, from the business process model. Other views will be also modeled as the dynamic view of services interaction specified by sequence diagrams, as defined in BPSOM. The sub-set of concepts and relationships that we have defined in the ontology and used in the QVT transformations definition are shown in Fig.4; the complete set of concepts and relations is detailed in [13].

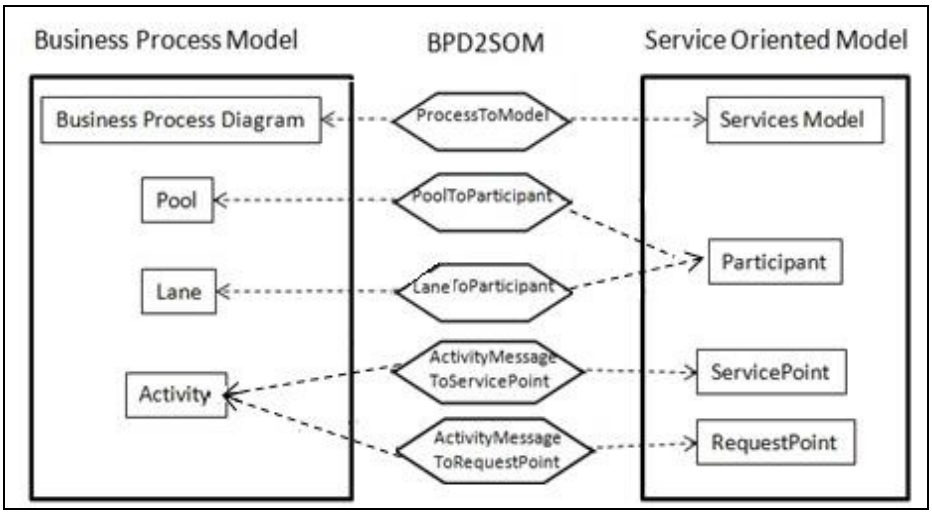

Fig. 4. Key concepts and relationships used in QVT transformations

First of all the services model is obtained from the business process diagram (BPD) as the root element. Each pool in the BPD defines a participant in the services architecture model, and each lane in each pool corresponds to an internal participant. Each participant provides and requires services corresponding to exchanged messages with other participants, so, the generated services will be associated with each participant. Each activity marked as "Service Type" will be transform into a ServicePoint or RequestPoint, depending on the direction of the message: if it is an incoming message then the service is provided in the generated ServicePoint, if it is an outgoing message then the service is request in the generated RequestPoint. We are aware that there are many other alternatives to specify these transformations, which we will explore in our future work. In Table 1 the QVT transformations from BPMN to SoaML comprising the concepts and relationships shown in Fig.4 are presented.

Table 1. Sub-set of the defined QVT transformations from BPMN to SoaML

Sub-set of relation rules defined
top relation ProcessToModel { checkonly domain bpmn bp : bpmn::BpmnDiagram{name = pn}; enforce domain soaml sm : SoaML::Model{name = pn }; }
top relation PoolToParticipant { checkonly domain bpmn p : bpmn::Pool{name = pn}; enforce domain soaml s : SoaML::Participant{name = pn};}

```
top relation LaneToParticipant {
   checkonly domain bpmn p : bpmn::Lane{name = pn };
   enforce domain soaml s : SoaML::Participant{ name = pn };}
top relation ActivityMessageToServicePoint {
   checkonly domain bpmn c : bpmn::Activity{lanes = p : bpmn::Lane{},
         activityType = bpmn::ActivityType::Task,
         incomingMessages = im : bpmn::MessagingEdge{},  name = cn};
  enforce domain soaml t : SoaML::ServicePoint {
         participant = s : SoaML::Participant {},
         isService = true,  name = cn};
         when { p.pool.bpmnDiagram.pools.lanes.activities -> exists
                (x:bpmn::MessageVertex | (x.outgoingMessages.target =
                c.incomingMessages.target) and
                (x.oclAsType(bpmn::Activity).activityType=c.activityType));
                PoolToParticipant (p.pool, s); }}
```

The first rule shown in Table 1 named "ProcessToModel" generates a SoaML "Model" element from a "BpmnDiagram" element corresponding to the BPD of BPMN, relating the top level business process to the services model. The second rule named "PoolToParticipant" generates SoaML "Participant" elements from "Pool" elements of the BP model, one Participant from each Pool. The third rule named "LaneToParticipant" generates SoaML "Participant" elements from "Lane" elements inside the pools of the BP model, which will be used to describe the internal architecture of each participant in the services architecture. The fourth rule named "ActivityMessageTo ServicePoint" generates SoaML "ServicePoint" elements from those activities in the BP model which have incoming messages from another activity, associating it with the participant that provide the service. In this rule it is necessary to look for all the activities in the BPMN model to see which are connected by messages connectors. The OCL expression in the "when" clause of the rule, evaluate all the activities in the model, checking whether they have outgoing messages to the activity being evaluated, that is incoming messages, and comparing the type of activities to be "Activity" instead of "Service" as we defined. This is a restriction of the BPMN Modeler metamodel in which activity elements are defined as "Activity", "Gateway", etc. and are differentiated with the ActivityType property, not providing the types "Service", "Manual", etc that we defined to use, which is minimal for the purpose of demonstrating the feasibility of QVT transformations from BPMN metamodel to SoaML metamodel.

4.2 QVT transformations example

To illustrate the proposal for SoaML service models generation from BPMN business process models, the Make Appointment business process from a generic hospital is modeled using the Eclipse BPMN Modeler, as shown in Fig. 5.

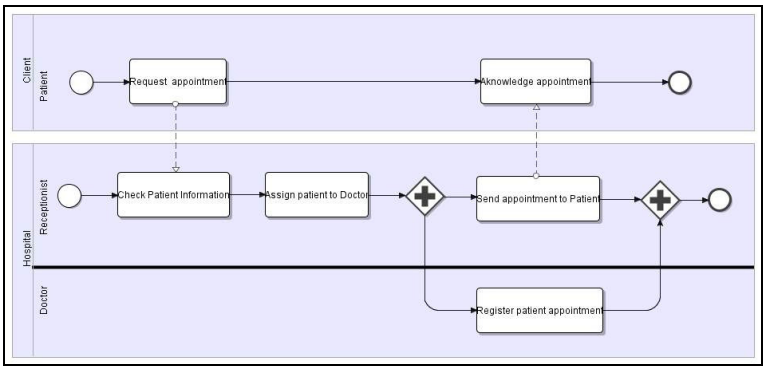

Fig. 5. Make Appointment business process

The business process shown in Fig. 5 starts when a Patient requests an appointment with a Doctor. The Receptionist checks the Patient information assigning a Doctor. After that, two activities are executed in parallel: the Doctor registers the appointment with the Patient and a communication is sent to the Patient with the appointment's information. From this business process we want to obtain two Participants: Client and Hospital, three Participants including in the previous ones: Patient for Client, and Receptionist and Doctor for Hospital. The Client participant offers a service in the activity "Acknowledge Appointment", and requires a service from the Hospital in the "Check Patient Information" activity. The Hospital participant offers a service in the activity "Check Patient Information", and requires a service in the activity "Acknowledge Appointment". After executing the defined QVT transformations the target XMI file is obtained with the SoaML elements generated, as shown in Fig. 6.

bpmn.ecore | SoaML.ecore | source.bpmn | source.bpmn_diagram | bpmn2soaml.qvt | target.bpmn

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SoaML="http://www.soaml.org/SoaML/1.0.0"
xsi:schemaLocation="http://www.soaml.org/SoaML/1.0.0 platform:/resource/BPMN2SOAML/metamodel/SoaML.ecore">
  <SoaML:Participant xmi:id="_n9TlkGcYEd-yM_IJR4RUBA" name="Patient"/>
  <SoaML:Participant xmi:id="_n9TlkmcYEd-yM_IJR4RUBA" name="Receptionist"/>
  <SoaML:Participant xmi:id="_n9TllGcYEd-yM_IJR4RUBA" name="Doctor"/>
  <SoaML:Participant xmi:id="_n9TllmcYEd-yM_IJR4RUBA" name="Client"/>
  <SoaML:ServicePoint xmi:id="_i08-WGddEd-yM_IJR4RUBA" name="Acknowledge Appointment" isService="true"
    participant="_i08-VmddEd-yM_IJR4RUBA"/>
  <SoaML:Participant xmi:id="_i08-WmddEd-yM_IJR4RUBA" name="Hospital"/>
  <SoaML:ServicePoint xmi:id="_i08-XGddEd-yM_IJR4RUBA" name="Check Patient Information" isService="true"
    participant="_i08-WmddEd-yM_IJR4RUBA"/>
  <SoaML:Model xmi:id="_i08-XmddEd-yM_IJR4RUBA" name="MakeAppointment"/>
</xmi:XMI>
```

Design | Source

Fig. 6. SoaML target XMI file generated from the BPMN business process

The two participants and the three included participants are generated; for the Client participant the ServicePoint "Acknowledge Appointment" is created, and for the Hospital participant the ServicePoint "Check Patient Information" is obtained. For the generation of the associated RequestPoint we are working on the rules to navigate through the messages to the target participants that provide them. After the SoaML XMI file is generated, it has to be loaded into the Magic Draw Cameo SOA+ plug-in to show the SoaML diagrams and to generate the code by the ModelPro engine.

5 Related work

In the last few years there have been many efforts aiming to apply SOC and MDD paradigms to business processes, from which a selection of works that defines methodological approaches or aims to automate the generation of service and software artifacts from business process are presented in the following. As the best of our knowledge there are no other works relating BPMN models with SoaML models directly as we do, but transformations from BPMN to UML can be seen in [33] where UML artifacts as Activity Diagrams (AD) and Collaboration and Deployment diagrams are generated from BPMN business process models, as a way to travel form business to IT vision, and [34] where BPMN business processes are transformed into AD and from these diagrams use cases and analysis classes are generated, with focus on business process security. Model driven approaches are proposed in [35] including a methodology, models, metamodels and transformations to obtain service oriented Web Information Systems (WIS) expressing the interaction of services to perform business processes, in [36] a business value model is integrated to the work to derive software artifacts from it using ATL[37], in [38] collaborative service oriented architecture is defined and transformations from BPMN models into UML models and BPEL models also using ATL. In [39] models and metamodels for services are defined in the PIM4SOA approach relating them to the defined architecture (brokerless, centralized and decentralized broker). Transformations based on the application of patterns are proposed in [40] starting with the macroflow-microflow pattern which establishes the conceptual basis and the process-based integration architecture pattern that guides the design of an architecture based on sub-layers for the service composition layer. In [41] conceptual transformations are defined based on the successively application of patterns from the top to the bottom layer, using graphs for pattern matching. [42] proposes going from BP models to technical processes matching existing services by applying transformation patterns classified with respect to the quality of the transformation. [43] goes from a business model (CIM) to an analysis model (PIM), identifying serviceAction in tasks, then to a Design model (Architecture specific model, ASM), mapping services to the target architecture. In [44] a methodology for the development of services associated with business processes is defined with focus on Web Services (WS) implementation, and in [45] a survey on existing approaches for identification and analysis of Business and Software Services along with a consolidated proposal is presented.

6 Conclusions and future work

The tool support presented in this article is part of the MINERVA framework we are working on, to support the continuous business process improvement based on the business process lifecycle for BPM activities. The described elements comprise a methodology BPSOM as a basis for the development process, QVT transformations to automate as much as possible the defined steps for services generation, and a set of integrated tools using as a basis the Eclipse open IDE. BPSOM is modeled in the EPF Composer as a method plug-in to be integrated in the existing software development

process used in the organization, with this tool. A proof of concept example was developed using the selected tools, to illustrate the steps defined in the proposal to go from a BPMN business process model to a SoaML service oriented model. The defined QVT transformations although simple serve as the basis for the further definition of transformations for the remaining SoaML elements. We believe that MINERVA could be a useful guide to be used in organizations that need rapid and easy integration of methodologies, tools and concepts to adopt the BPM, SOC and MDD paradigms. Our future work is now focused on the definition of the remaining QVT transformations to effectively generate complete SoaML services models from BPMN business processes models, from which to obtain the associated code. We are also working in an implementation of the SoaML profile as an Eclipse plug-in, and we plan to integrate the method plug-in BPSOM into Eclipse to automatically aid in the realization of the methodology activities' flow.

Acknowledgments. This work has been partially funded by the Agencia Nacional de Investigación e Innovación (ANII) from Uruguay; ALTAMIRA project (Junta de Comunidades de Castilla-La Mancha, Spain, Fondo Social Europeo, PII2I09-0106-2463) and PEGASO/MAGO project (Ministerio de Ciencia e Innovacion MICINN, Spain, and Fondo Europeo de Desarrollo Regional FEDER, TIN2009-13718-C02-01).

References

1. Business Process Management Initiative, <http://www.bpmi.org/>
2. Smith,H.,Fingar,P.,Business Process Management: The third wave, Meghan-Kieffer, (2003)
3. Papazoglou, M.; Traverso, P.; Dustdar, S.; Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenge, IEEE Computer Society, (2007)
4. Krafzig, D. Banke, K. Slama, D., Enterprise SOA, Service Oriented Architecture: Best Practices, Prentice Hall, 1st. ed., (2005)
5. Erl, T., SOA: Concepts, Technology, and Design,Prentice Hall, (2005)
6. Mellor, S., Clark, A., Futagami, T., Model Driven Development - Guest editors introduction, IEEE Computer Society, September/October (2003).
7. Stahl, T.; Volter, M. et. al.: Model-Driven Software Development, Technology, Engineering, Management, John Wiley & Sons, Ltd., (2006)
8. Model Driven Architecture (MDA) v. 1.0.1, OMG, http://www.omg.org/mda, (2003)
9. Weske, M., BPM Concepts, Languages, Architectures, Springer, (2007)
10. Mendling, J., Metrics for process models, Springer, (2008)
11. van der Aalst, W.M.P., ter Hofstede, A., Weske, M., Business Process Management: A Survey, In: International Conference on Business Process Management, (2003)
12.Delgado, A., Ruiz F., García - Rodríguez de Guzmán I., Piattini M., MINERVA: Model drIveN and sErvice oRiented framework for the continuous business processes improVement & relAted tools, In: 5th International Workshop on Engineering Service-Oriented Applications (WESOA'09), Stockholm, (2009), Springer (2010) in press.
13. Delgado, A., Ruiz, F., García - Rodríguez de Guzmán, I., Piattini, M.: Towards an ontology for service oriented modeling supporting business processes, In IV International Conference on Research Challenges in Information Science (RCIS'10), Niza, (2010)
14. Delgado, A., Ruiz, F., García - Rodríguez de Guzmán, I., Piattini, M.: Towards a Service-Oriented and Model-Driven framework with business processes as first-class citizens, In: 2nd Int. Conf. on Business Process and Services Computing (BPSC'09), Leipzig, (2009)

15. Business Process Maturity Model (BPMM), OMG, http://www.omg.org/spec/BPMM
16.Sánchez, L., Delgado, A., Ruiz, F., García, F., Piattini, M.: Measurement and Maturity of Business Processes. Eds.: Cardoso, J., van der Aalst, W.,Handbook of Research on Business Process Modeling, Information Science Reference (IGI Global),pp.532-556, (2009)
17. Eclipse Process Framework Composer (EPF Composer), http://www.eclipse.org/epf/
18.Query/Views/Transformations(QVT),v.1.0,OMG,http://www.omg.org/spec/QVT/1.0,(2008)
19. Business Process Modeling Notation (BPMN), OMG, http://www.omg.org/spec/BPMN/
20 Soa Modeling Language (SoaML),OMG, http://www.omg.org/spec/SoaML/, (2009)
21. BP Service Oriented Methodology (BPSOM) http://alarcos.esi.uclm.es/MINERVA/BPSOM/
22.Jacobson, I., Booch, G., Rumbaugh, J. The Unified Software Development Process, Addison-Wesley, (1999)
23. SOA Tools Platform (STP) BPMN Modeler, http://www.eclipse.org/bpmn/
24. Medini QVT, ikv++ technlogies ag, http://projects.ikv.de/qvt/
25. Magic Draw CameoSOA+,http://www.nomagic.com/ text.php?lang=2&item=338&arg=295
26. ModelPro, http://modeldriven.org/
27. XML Process Definition Language (XPDL), v.2.1, WfMC, http://www.wfmc.org/xpdl.html
28.WS BP Execution Language (WS-BPEL),OASIS, http://docs.oasisopen.org/wsbpel/2.0/
29.BPModeling Notation (BPMN), v.2.0, OMG, http://www.omg.org/spec/ BPMN/2.0/, (2009)
30. van der Aalst, W.M.P., Reijers, H. A., Medeiros, A.,Business Process Mining: an Industrial Application, Information Systems Vol.32 Issue 5, 713-732, (2007)
31.ProM, Process Mining Group, Eindhoven University of Technology, Eindhoven, The Netherlands, http://prom.win.tue.nl/research/wiki
32. SoaML implementations, http://www.omgwiki.org/SoaML/ doku.php?id=tool_support
33.Liew,P., Kontogiannis,K. Tong,T., A Framework for Business Model Driven Development,12th Int. Workshop on Se Tech. and Engineering Practice (STEP'04), (2004)
34.Rodríguez,A.; Fernández-Medina, E.; Piattini, M.: Towards CIM to PIM Transformation: From Secure Business Processes Defined in BPMN to Use-Cases. 5th Int. Conf. on BPM (BPM'07)(2007)
35. de Castro, V., Marcos, E., López Sanz, M., A model driven method for service composition modelling: a case study, Int. J. Web Engineering and Technology, Vol. 2, No. 4, (2006)
36.de Castro V., Vara Mesa J. M., Herrmann E., Marcos E., A Model Driven Approach for the Alignment of Business and Information Systems Models, (2008)
37. Jouault, F., Kurtev, I.,Transforming Models with ATL (ATLAS Transformation Language), Satellite Events at the MoDELS Conference, (2005)
38. Touzi J., Benaben F., Pingaud H., Lorré J.P., A model-driven approach for collaborative service-oriented architecture design, Int. Journal of Prod. Economics,Vol.121 Is.1, (2009)
39. Roser,S., Bauer,B., Muller,J., Model- and Architecture-Driven Development in the Context of Cross-Enterprise Business Process Engineering", International Conference on Services Computing (SCC'06), (2006)
40.Zdun, U., Hentrich, C., Dustdar, S., Modeling Process-Driven and SOA Using Patterns and Pattern Primitives, ACM Transactions on the Web, Vol. 1, No. 3, Article 14, (2007)
41. Gacitua-Decar V., Pahl C., Pattern-based business-driven analysis and design of service architectures, 3rd Int. Conf. on Software and Data Technologies SE (ICSOFT'08), (2008)
42. Henkel,M., Zdravkovic,J., Supporting Development and Evolution of Service-based Processes, International Conference on e-Business Engineering (ICEBE'05), (2005)
43. Herold S., Rausch A., Bosl A., Ebell J., Linsmeier C., Peters D., A Seamless Modeling Approach for Service-Oriented Information Systems, 5th International Conference on Information Technology:New Generations(ITNG 08), (2008)
44.Papazoglou, M., van den Heuvel, W., Service-oriented design and development methodology, Int. J. Web Engineering and Technology, Vol. 2, No. 4, pp.412-462, (2006)
45. Kohlborn T., Korthaus A., Chan T., Rosemann M., Identification and Analysis of Business and SE Services- A Consolidated Approach, IEEE Transactions on Services Comp., (2009)

OASEF: A Synthetic Approach to Service Engineering

Yuanzhi Wang

Department of Computer Science
The Australian National University, Australia
derek.wang@anu.edu.au

Abstract. Engineering complex service-oriented systems presents grand challenges due to their great complexity, volatility, and uncertainty in their rapidly evolving technology and social contexts. It demands an effective engineering approach in order to satisfy the needs of service economics. This paper proposes an approach called Organic Aggregation Service Engineering Framework (OASEF). Experience from proof-of-concepts case studies shows that it provides a practical means to develop service-oriented systems. It enables and promotes a focus on higher-level intellectual engineering efforts, and provides a mechanism to capture and reuse engineering capacities in a model-driven environment.

Keywords: Service Engineering, Model-driven Engineering, Service-oriented Computing

1 Introduction

Deriving and managing complex Service-Oriented Systems (SOS) presents great challenges due to their nature and characteristics in a dynamically complex environments [1]. Firstly, SOS often involve great complexity in terms of variety, scope, and inter-relationship. On the one hand, technologies such as Service-Oriented Computing (SOC) and Cloud Computing (CC) greatly facilitate development and integration of applications and systems. On the other hand, economical and social globalisation processes inevitably force individuals, organisations, and communities to collaborate and compete with each other within interacting value-chains. As a result, the size, variety, and scope of interrelated systems scales up rapidly.

Secondly, SOS often present a high degree of uncertainty. Autonomous services are often provided and consumed by different agents for different types of purposes, without necessarily knowing each other in advance [2]. Moreover, defining a service contract at any time cannot completely incorporate unknown usage with necessary variations. Therefore, it presents great challenges to satisfy not only the explicit requirements of current target applications, but also the needs of envisioned future applications or unknown potential users. Lastly, in highly volatile and heterogeneous environments, the velocity and variety of

appropriate response have to match those of the environmental changes. An effective service engineering hence has to support and facilitate dynamic evolution by changing and reshaping their internal structures and behaviour, in a more frequent and predictable fashion, compared with traditional systems.

These special characteristics and challenges can hardly be resolved by merely adopting traditional software processes and engineering methodologies, or solely relying on more advanced implementation technologies [3]. In order to enable incorporation of wide range of business and social systems within rapidly growing global service economics, there is an important and urgent demand for a mature discipline of service engineering [4]. In our view, such a discipline, to be effective, must provide sufficient support for addressing the grand challenges of complexity, uncertainty, and volatility while engineering SOS in their open environments. We believe one way to approximate this ideal is to enable and promote derivation and reuse of important higher-order intellectual efforts and lower level technological capacities. This is because evidences have shown that great complexity involved in engineering problems is better dealt with by intellectual cognitive experience and capacities, which lead to sensible perception of reality, logical reasoning of problematic situations, and systematic derivation of conceptual plans [5]. Therefore, effective and efficient accumulation and use of these intellectual resources is the key crucial factor to manage the ever-presenting complexity, uncertainty, and volatility in a timely fashion while changes present themselves. In the meantime, in order to take full advantage of advanced technologies and implementation infrastructures, and at the same time, deal with their increasing heterogeneity and complexity, a discipline of service engineering should also exploit effective means to develop, encapsulate, and automate the engineering capacities and processes about lower-level realisation and implementation.

Therefore, our vision of an effective service engineering approach should enable and support efficient and flexible formation and exploitation of both higher-order intellectual resources and lower-order implementation processes. That is, both types of engineering resources should be explicitly identified, developed, captured, and used, as major engineering means, to produce and manage systems, in a flexible, systematic, and automatic means, at least partially if processing in its entirety is not possible. Specifically, our objective is to achieve an effective service engineering approach that: i) promotes a focus on high-level intellectual activities within the world of human mind, such as exploring and understanding problematic situations, and identifying and capturing higher order engineering purposes and intentions; ii) links the captured higher-order engineering resources with other lower-order engineering activities by using the former to guide and shape the latter systematically and sensibly, within a coherent overall process; and iii) provides a means to capture, aggregate, and reuse these important engineering resources, including both higher-order and lower-order capacities, to facilitate flexible and rapid aggregations of processes and systems.

This paper presents a new approach to service engineering based on such a vision, which is called Organic Aggregation Service Engineering Framework

(OASEF). It is based on a philosophy that takes a synthetic approach to growing and managing processes and systems via sensible and responsive aggregations of resources and capacities. This work mainly has the following contribution: firstly, it emphasises exploration and exploitation of higher-order engineering efforts, and links them with lower-level technological resources within an overall process model; secondly, it promotes a concept of *organic* aggregation in engineering that captures engineering capacities as reusable resources, and, more importantly, uses them in a sensible, agile, and controlled fashion.

The remainder of the paper is organised as follows: section 2 presents the proposed OASEF framework in details, including a general engineering framework it conforms to, its conceptual model, and modelling methodology; section 3 briefly illustrates its important concepts and methods using results from two proof-of-concept case studies; section 4 analyses its features and limitations based on observation and experience during the case studies, and compares it with some related work; section 5 concludes this paper with a summary.

2 The OASEF approach to service engineering

It is important for a discipline of service engineering to have a well-founded engineering framework that coherently organises engineering activities and resources according to the nature and characteristics of services and systems under consideration. The aim of this work is to provide and assess such a framework.

2.1 OASEF conceptual model

The design of OASEF started from generalisation of other engineering disciplines. Figure 1 depicts a general conceptual framework to which OASEF conforms. It includes a theoretical foundation, or a coherent conceptual system, which consists of of inter-related theories, knowledge, and wisdom that are shaped or influenced by individual and social experience, beliefs, philosophies, and culture [6]. Such a foundation, either implicitly or explicitly, guides and shapes, positive experience and practices that are justified in the course of continuous engineering activities, which gradually form a range of concrete and specific guiding principles. These principles, articulated, well explained, and understood, provide valuable guidance for engineering activities in practice.

Moreover, based on the abstract foundation and principles, more concrete engineering activities are arranged and conducted within a specific engineering process, which, in turn, is realised or supported by a range of practical engineering methodologies. The latter provides specific and concrete means to realise the aiming higher level engineering purposes, and to solve practical human problems in a controllable, repeatable, and efficient fashion [6]. Specific languages, tools, and implementation techniques practically enable or facilitate the engineering processes and methodologies, and eventually produce or manage the target systems in reality. Altogether, these coherent engineering elements,

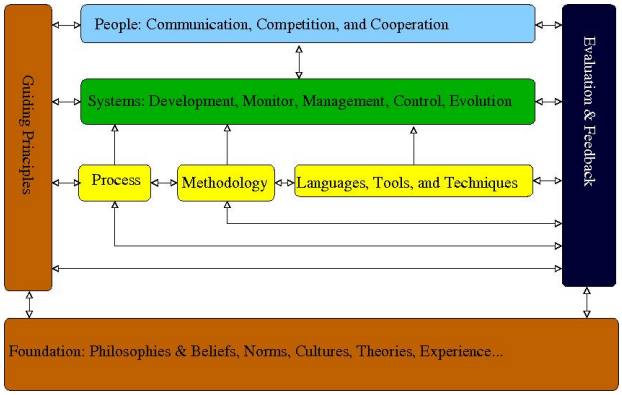

Fig. 1: A meta-model for service engineering

involving a variety of stakeholders, contribute to engineering of resources, infrastructure, events, processes, and systems that aim to satisfy identified engineering needs and human purposes.

The overall structure of OASEF is depicted in figure 2. It conforms to the general framework, and is based on a theoretical foundation rooted in some multi-disciplinary knowledge, such as general system theory and modern philosophy of mind. It also incorporates some practical principles by which its process and methodology are guided or reinforced. These guiding principles, such as "environment-driven view of service" [7], "pervasive change and evolution", "support for systems adaptation and agility", and "facilitating knowledge and capability reuse", will not be presented here due to space limitation. Although their meanings are well known, widely accepted, and often taken for granted, they provide important empirically verified guidance for engineering activities.

As shown in the figure, OASEF incorporates a flexible process model called Organic Aggregation Process (OAP) in the context of service engineering. Figure 3 illustrates its inner structure and inter-relationships at a detailed level. The lines with text besides them represent specific process activities, whereas the joint point between two lines represents direct correspondence between one activity to another. Moreover, activities are arranged hierarchically in this structure. An activity represented by a single line, starting from point A to point B in the clockwise direction, may comprise a sequence of subordinate activities that are represented by a series of adjacent lines, which also starts from point A and ends with point B clockwise. Superordinate activities are represented by thicker lines and larger-size fonts on the hierarchy. For example, *perception* comprises subordinate activities of *sensation* and *abstraction*. The latter, in turn, comprises *system abstraction* and *general abstraction*.

The OAP concepts and process are integrated in OASEF at various level, as illustrated in the middle part of figure 2. The activity of *Perception* acquires information about the *Reality* world through *Sensation*, and forms general or system-specific abstraction of knowledge. The latter is used by *Conception* to conduct human intellectual activities in the world of *Mind*. Specifically, *Concep-*

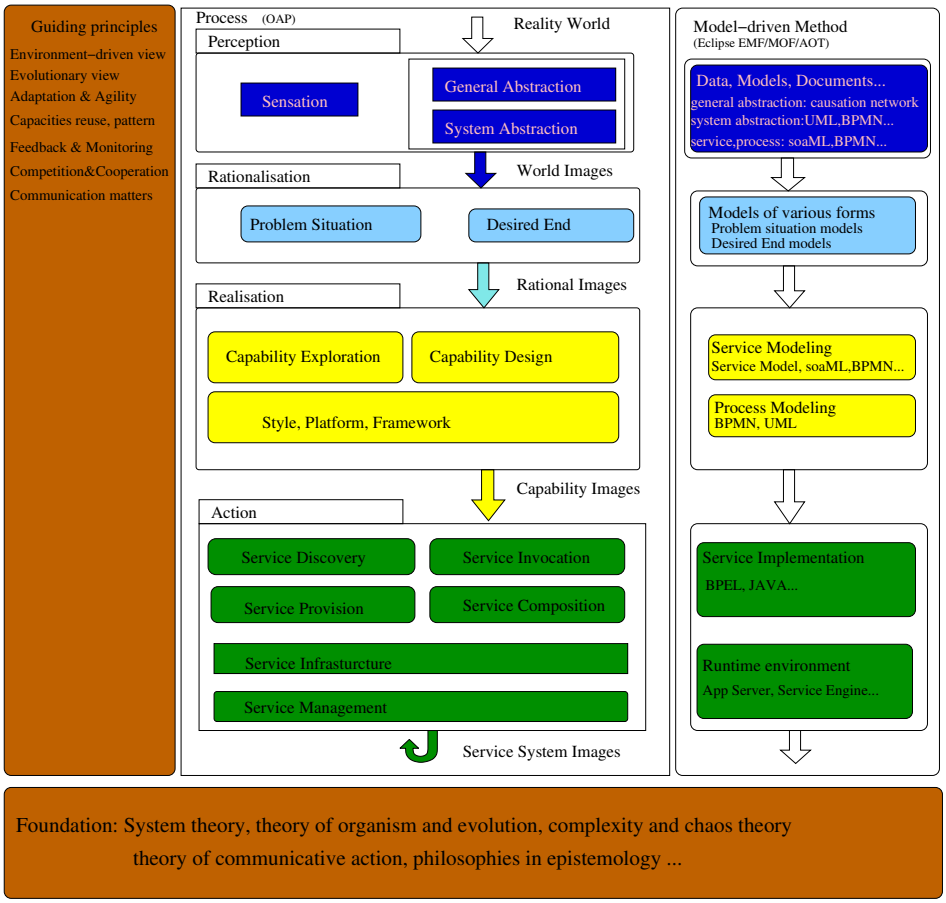

Fig. 2: OASEF: Organic Aggregation Service Engineering Framework

tion activities involve *Rationalisation*, the reasoning and sense-making processes that produce understanding of perceived situations, inferred problems, and desired ends, which are realised by subordinate activities such as *Problem Situation* and *Desired End*. *Conception* also includes high-level design, engineering decision-making, and plan-making sub-processes through *Realisation* activities, such as *Capability Exploration* and *Capability Design* that identify and design desired capabilities in accordance with outcome of *Rationalisation*. Furthermore *Action*, conducted in the world of *Reality*, deal with concrete service systems by finding and invoking existing services, providing new services, or composing composite services. *Service Infrastructure* provides information of implementation environment such as Enterprise Service Bus (ESB), whereas *Service Management* provides control and monitoring of services in run-time environments.

Therefore, OASEF derives and manages services using interconnected OAP activities. Achievement of these activities collectively forms specific images representing unified repository of knowledge, which can be referenced, or used as

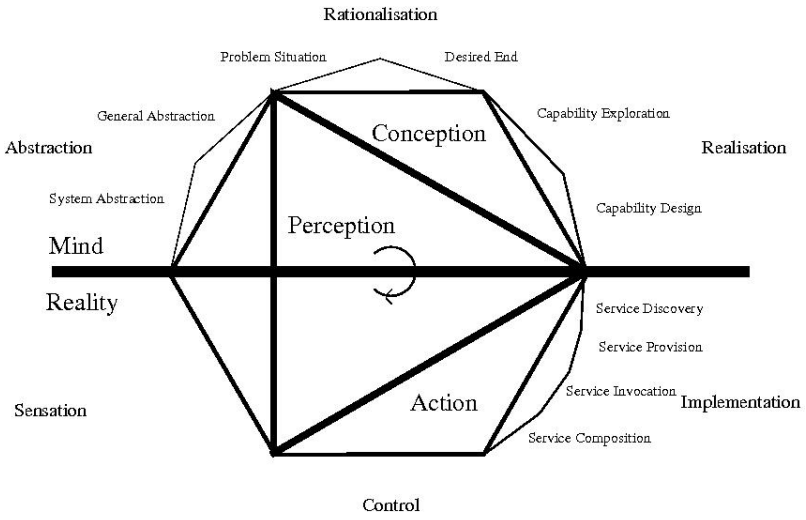

Fig. 3: The structure of OAP process model in service engineering

input, by successive activities. For example, the *Rationalisation* takes *World Images* from *Perception* as input. It produces *Rational Images* that represent rationalised *Desired Ends* models to deal with identified *Problem Situation*. *Realisation* takes the *Rational Images* and produces *Capability Images* that represent required realisable capabilities. The latter is taken by *Action* as input and eventually produces SOS that alter the state of the *Reality* with an aim of improving rationalised problematic situations.

2.2 OASEF modelling and tools support

The conceptual model of OASEF is brought into reality using specific modelling techniques and model-driven methodologies. Various abstract models, either general purpose modelling language such as UML, or specially designed modelling languages, are used as essential artefacts to facilitate OASEF activities, and to link these activities altogether throughout the engineering life cycle.

OASEF emphasises activities of *Abstraction* and *Rationalisation* in the early stage of the life cycle. The objective of *Rationalisation* modelling is to analysis and justify the rationale and needs of engineering processes toward sensible decisions of actions. Two types of modelling formalism, namely Problem Situation Models (PSM) and Desired End Models (DEM), are designed to capture the crucial higher order cognitive achievement. For example, output of *Problem Situation* activity is captured in terms of complex inter-relationships between various problems, facts and constraints, and high-level purposes. Together, PSM and DEM provide an important means to explore and represent problematic situations and desired ends that justify successive engineering activities and their produced engineering results.

As an example, the graphical notations of *DEM* are depicted in figure 4. Coloured rectangles with names inside represent various desired ends elements. They capture the engineering intentions in terms of their concreteness and scope. For example a *Goal* is more specific in scope, less abstract to define and communicate, and easier to measure, compared with *Objective* and *Ideal*. Specifically, light yellow rectangles represent *Ideal*, whereas light green and dark green ones represent *Objective* and *Goal* respectively. Moreover, white rectangles are used to represent higher order *Capabilities* that provide contextual meaning and desired value in support of service identification and realisation. Relationships among various DEM elements are represented by connecting arrowed lines. For example, an arrowed line can link a *Ideal* with its subordinate *Objective*, a *Objective* with composing *Goal*, or a *Goal* with its subordinate one.

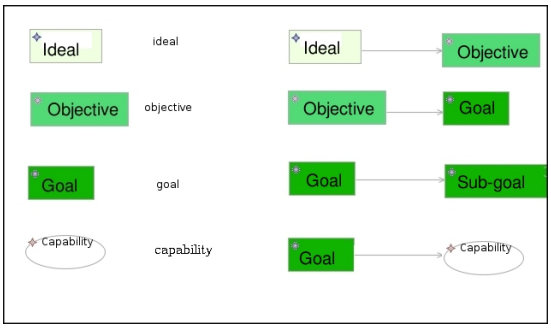

Fig. 4: Graphic notations of DEM

Every OASEF activities produce either domain-specific or general-purpose models that all conform to a unified meta-meta-models. Therefore, activities in OASEF can be conducted and inter-connected in a coherent fashion by manipulating and utilising every model in the same way. For example, UML models based on Eclipse Modelling Framework (EMF) are used to capture information and knowledge in *Sensation* and *Abstraction*, in the same way PSM and DEM are manipulated. UML activity diagram, soaML, State Machine, and BPMN are used in *Realisation* similarly. Some models transformation techniques on Eclipse modelling platform are also integrated in OASEF to transform various OASEF models into different forms and generate desired system in an automatic or semi-automatic fashion.

Moreover, OASEF provides a mechanism called *Epitome* to reuse engineering capacities to conduct OAP activities. It is a typical and justified means, or capability to achieve some engineering purposes. It is generalised from proven examples that tightly link two OAP activities together, such as a specific *Realisation* that is able to achieve predictable and optimised results in reality to improve identified problematic situations. Applying an existing *Epitome* directly produce a previously proven outcome without having to go through the particular activities. For instance, it generate specific *Realisation* models that are able to improve the typical problems.

A number of supporting tools are integrated within an integrated supporting tools environment, called IPEOAP. It is developed using Java programming language on top of Eclipse IDE, EMF and modelling platform. A range of graphical model editors are developed to view, create and modify models for various OAP activities, such as PSM and DEM.

3 Case studies

Two controlled case studies were conducted as proof-of-concept, as oppose to proof-of-performance, which aim to assess whether, in general, its objectives are meet in real world settings. The first case study is in the context of online travel booking business, a typical scenario used in service community. While the second one is based on Australian First Home Saver Accounts (FHSA) scheme that was introduced by the federal government in 2008 to help residents to purchase their first homes. This section briefly presents some outcomes of these two case studies to illustrate the main feature, concepts and application of OASEF.

A PSM in the context of online travel booking exemplifies the high level analysis of *Problem Situation* in *Rationalisation*. It contains many higher level *Purposes* for online booking, such as "Ease of use", "Quick Responsiveness", "Reliable booking", "Economic price", and "Secure and trust". These PSM elements are generated systematically in accordance with general knowledge captured in *general abstraction* models within *world images*. It also contains a range of higher level *Problems* such as "The booking process takes too long", which violates the "Quick Responsiveness" *Purpose*. This *Problem* is also caused by other *Problems* such as "Too many service providers involved", "Some providers are less efficient than others", "Insufficient network bandwidth", "Diversity in interface and interaction mechanisms", and "The ordering process is too complex" that is caused by "sequential correspondence". Exploration of problems and purposes reveal their nature and relationships and help to identify, understand, and capture important problematic situations.

The PSM also help to systematically generate other models during successive engineering activities. As an example, since the problem "Online travel booking takes too long time" is identified as a major problem that affects a higher priority purpose "Quick Responsiveness", a DEM, depicted by figure 5, is created to reveal the best desired improvements to address the problem. The construction process of DEM is guided by, and makes referenced to, elements in the above PSM. For example, The desired improvement of efficiency during travel booking process includes a higher level *ideal*, called "Fast booking", which is achieved via a number of more specific *Objectives*, such as "Service provider filtering", "Caching", "Simplify booking process", and "Parallel processing", which, respectively, filters out slow service providers, provides cache to provide repeated information locally, simplifies the booking processes, and interacts with service providers in a parallel and asynchronous fashion. The last *Objective* contains some lower-level specific *Goal*s, such as "Parallel booking" and "Parallel queries". The identification of these improvement and desired ends at different

level of details helps to discover the "right" and achievable goals and desired capabilities to address the important problems.

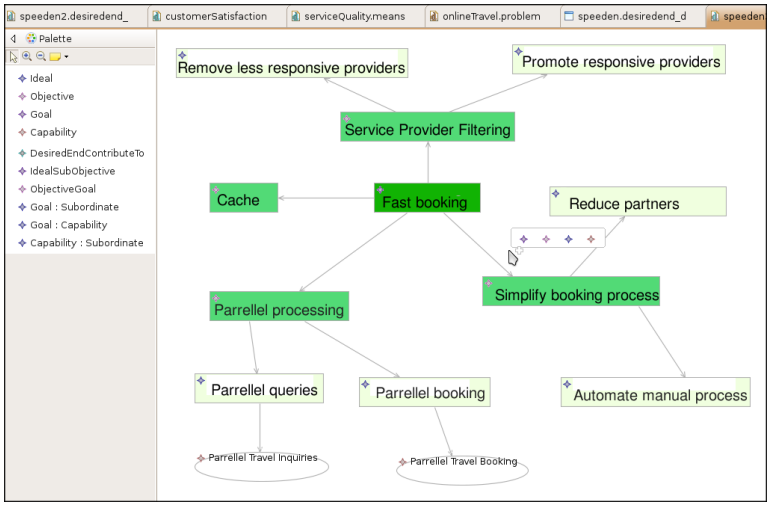

Fig. 5: An example of DEM in an online travel booking scenario

In OASEF, higher level models such as *System Abstraction*, PSM, and DEM are used to derive and capture intellectual achievement as crucial engineering resources. In the meantime, more concrete design is also captured by models. For example, UML Activity diagrams are used to describe the internal structure and behaviour of subordinate business processes in *Capability Design*, with the aim to achieve desired ends in DEM. A design of a process to "Close an existing FHSA account" is systematically derived in accordance with the identified problem of "Cannot manage FSHS account in current bank system" in its PSM. It contains interconnected required capabilities within an orchestrated structure, such as "Check Eligibility", "Acquire state information", "Validate Customer Closure Form", "Fund processing", and "Notify Customer Result". When conducting *Service Composition* during *Implementation*, the above *Capability Design* model is automatically transformed into more concrete formalism, such as Business Process Execution Language(BPEL) models, which can be directly executed in a run-time environment, such as Apache ODE BPEL engine used in the case studies. The transformation process from UML Activity Diagrams to BPEL can utilise various BPEL transformation tools such as the MDD4SOA tools set [8].

4 Evaluation and analysis

Some observation and assessment of OASEF are made during the design and implementation of proof-of-concept systems in real world settings. Due to a focus on, and methodological support for, higher-level intellectual efforts, OASEF helps to analyse and understand a complex situation, and to systematically derive sensible and rational business decisions, for example, in one of the case

study, whether or not provide support for a bank business under specific legislation environment. The modelling mechanism and tool environment enables developers to concentrate on exploring and understanding higher order business issues including complex situations and desired business ends to be achieve, using models such as PSM and DEM. The graphical tools provided in IPEOAP facilitate the manipulation of these higher-level intellectual efforts, such as design and manipulation of *Abstraction*, *Problem Situation*, and *Desired End*. The use of unified EMF modelling meta-model enables interactions and cross-references among various OASEF activities, using either general purpose UML models or specially designed models such as DEM.

In the meantime, "accidental complexity" of underlying implementation technologies is largely hidden during the case study due to capturing and exploitation of engineering capacities by using *Epitomes* and automatic model transformation techniques. For example, the processes to create database persistence logic, web service provision and invocation, and web user interface are completely captured in various *epitomes*. Consequently, given a *System Abstraction* model such as information model for a flight or bank account, OASEF enables 100 percent automatic code generation, which produces systems capable of collecting information from users, persisting data, and providing relevant information service. Experience from the case studies shows that, based on well captured higher order models and technological capacities, generation of concrete systems at the implementation level is relatively fast, and requires little human manipulation.

Some issues and limitations are also revealed by the case studies. Although the proof-of-concept case studies were based on real world settings, the implemented systems are not assessed in real business environment. In fact, since the controlled case studies were designed to demonstrate and evaluate the objectives of OASEF, by the same person who designed the framework, the results are hence less convincing compared with empirical practice by third parties. Moreover, various OASEF activities are neither monitored nor validated in the engineering process, which makes it hard to identify problems when things go wrong. Furthermore, due to its proof-of-concept purpose, these case studies did not cover all aspects of the framework such as *sensation* and *control*. It hence requires further work to improve the prototype and conduct thorough empirical evaluation in practice, such as applying evaluation metrics to quantitatively assess the effectiveness in real world projects, ideally on a benchmark system, and in comparison with other approaches.

Based on the empirical experience from the case studies, OASEF is also compared with a range of other approaches, which have varying focus, objectives, and realisation methods. Kohlborn et. al. proposed a consolidated approach that aims to provide a good business and IT alignment by layering them separately with certain linkage in between [9]. For each layer, four stages, namely *Preparation*, *Identification*, *Detailing*, and *Prioritisation*, are used to progressively identify, elaborate, and provide desired services that, collectively, form the systems. However, the nature and content of its higher order activities such as "Conducting interviews", "Conducting capability analysis", and "Defining domains" are

vaguely defined and lack practical guidance. Moreover, this work provides insufficient supports for capturing and managing both higher level business and lower level technologies in a flexible fashion within its predefined layers. Its strength is weakened in practice due to the lack of concrete formalism and modelling techniques. In comparison, OASEF, founded on sounds theoretical base, defines the scope, purpose, and relationship of its activities, and more importantly, provides specific modelling mechanism to manage them.

Lamparter and Sure also proposed an interdisciplinary methodology that combines a Web Service engineering method with market engineering and ontology that aims to coordinate services and customer in a collaborative environment [10]. Although it covers a full range of system analysis and design activities, no specific means is provided to manage uncertainty and volatility in a dynamic environment, which makes it less effective when changes are required in a timely fashion. Whereas OASEF attacks this issue by allowing flexible aggregations of previously-proven engineering capacities and process automation in a unified environment, in accordance with captured higher order rational justifications.

There are also a number of other model-driven approaches to service engineering [11, 8] that provides various modelling facilities and tools to map business processes into executable SOS. For example, a model driven technique is applied to build SOS by converting higher level business processes, captured in UML activity diagrams, into executable BPEL files [8]. Some tools are developed to facilitate reasonably smooth transition from business design captured in BPMN and activity diagrams, into system implementation [11, 8]. However, unlike OASEF, these approaches do not focus on, and provides no support for, some higher level intellectual efforts such as identification of problems and improvement. Although our previous work [7] also applied a model-driven approach to facilitate higher level modelling through conceptual *orientation* and *decision*, it does not realise our vision of service engineering due to some great difficulties, such as a lack of clear definitions and concrete formalisms for problem-level abstraction, and inadequate separation of various engineering concerns during each phase.

5 Conclusion

In this paper, we propose a synthetic approach to service engineering. It incorporates some inter-related elements including a conceptual foundation and guiding principles, a novel process model, a model-driven method, and an integrated supporting environment.

Two proof-of-concept case studies were conducted in real world settings, and are briefly presented in this paper to illustrate some important concepts and techniques. The results show that this approach can be applied in real-world settings to facilitate service engineering and achieve its design goals in general. More specifically, the following positive results are observed. Firstly, by using higher order models such as PSM and DEM, OASEF enables and promotes coherent higher order intellectual activities, by which effective services

and systems are presupposed. Models in *Abstraction*, *Rationalisation*, and *Realisation*, are captured as important engineering resources that can be located and aggregated together. They hence form important human intellectual assets that provide creative and valuable essence to achieve the "right" and optimised systems. Secondly, activities involving implementation technologies in *Action*, are also captured as reusable engineering capacities, and are used to automatically realise identified desired ends in a relatively easy way. Thirdly, using a model-driven method, OASEF creates an effective linkage between higher order intellectual efforts and lower level implementation processes, since both type of resources are organised in a unified, identifiable, reusable fashion. The coherently aggregation of resources and capacities are used altogether to systemically and automatically drive the engineering process to produce desired SOS.

Some important issues and limitations are also revealed during the prototyping and evaluation processing. Corresponding improvements and more thorough evaluation of its practical effectiveness in real world projects are essential to consider in future research.

References

1. Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F., Kramer, B.: Service-oriented computing: A research roadmap. Service Oriented Computing (SOC) (2006)
2. Chang, S.H., Kim, S.D.: A service-oriented analysis and design approach to developing adaptable services. Services Computing, 2007. SCC 2007. IEEE International Conference on (2007) 204–211
3. Dijkstra, E.: The humble programmer, acm turing lecture 1972. Communications of the ACM **15** (1972) 859–66
4. Cardoso, J., Voigt, K., Winkler, M.: Service engineering for the internet of services. Enterprise Information Systems (2008) 15–27
5. Ackoff, R.L.: Ackoff's best, his classic writings on management. John Wiley & Sons, New York (1999)
6. Checkland, P.: Systems thinking, systems practice. J. Wiley, New York (1981)
7. Wang, Y., Taylor, K.: A model-driven approach to service composition. In: Service-Oriented System Engineering, 2008. SOSE '08. IEEE International Symposium on. (2008) 8–13
8. Mayer, P., Schroeder, A., Koch, N.: A model-driven approach to service orchestration. In: IEEE International Conference on Services Computing, 2008. SCC'08. Volume 2. (2008)
9. Kohlborn, T., Korthaus, A., Chan, T., Rosemann, M.: Identification and Analysis of Business and Software ServicesA Consolidated Approach. IEEE Transactions on Services Computing (2009) 50–64
10. Lamparter, S., Sure, Y.: An interdisciplinary methodology for building service-oriented systems on the web. In: Services Computing, 2008. SCC '08. IEEE International Conference on. Volume 2. (2008) 475–478
11. Brambilla, M., Dosmi, M., Fraternali, P.: Model-Driven Engineering of Service Orchestrations. In: Proceedings of the 2009 Congress on Services-I-Volume 00, IEEE Computer Society (2009) 562–569

Inference of performance annotations in Web Service composition models

Antonio García-Domínguez[1], Inmaculada Medina-Bulo[1], and Mariano Marcos-Bárcena[2]

[1]Department of Computer Languages and Systems, University of Cádiz, C/Chile 1, CP 11003, Cádiz
{antonio.garciadominguez,inmaculada.medina}@uca.es
[2]Department of Mechanical Engineering and Industrial Design, University of Cádiz, C/Chile 1, CP 11003, Cádiz
mariano.marcos@uca.es

Abstract. High-quality services must keep working reliably and efficiently, and service compositions are no exception. As they integrate several internal and external services over the network, they need to be carefully designed to meet their performance requirements. Current approaches assist developers in estimating whether the selected services can fulfill those requirements. However, they do not help developers define requirements for services lacking performance constraints and historical data. Manually estimating these constraints is a time-consuming process which might underestimate or overestimate the required performance, incurring in additional costs. This work presents the first version of two algorithms which infer the missing performance constraints from a service composition model. The algorithms are designed to spread equally the load across the composition according to the probability and frequency each service is invoked, and to check the consistency of the performance constraints of each service with those of the composition.

Keywords: service level agreement, load testing, UML activity diagrams, service oriented architecture, service compositions.

1 Introduction

Service-oriented architectures have been identified as an effective method to reduce costs and increase flexibility in IT [1]. Information is shared across the entire organization and beyond it as a collection of services, managed according to business needs and usually implemented as *Web Services* (WS).

It is often required to join several WS into a single WS with more functionality, known as a *service composition*. Workflow languages such as WS-BPEL 2.0 [2] or BPMN [3] have been specifically designed for this, letting the user specify the composition as a graph of activities.

Just as any other software, service compositions need to produce the required results in a reasonable time. This is complicated by the fact that service compositions depend on both externally and internally developed services and the

network. For this reason, there has been considerable work in estimating the quality of service (QoS) of a service composition from its tasks' QoS [4,5] and selecting the combination of services to be used [6].

However, these approaches assume that the services already exist and include annotations with their expected performance. For this reason, they are well suited with bottom-up approaches to developing service compositions. On the other hand, they do not fit well in a top-down approach, where the user defines the composition and its target QoS *before* that of the services: some of them might not even be implemented yet. Historical data and formal service level agreements will not be available for these, and for new problem domains, the designer might not have enough experience to accurately estimate their QoS.

Inaccurate QoS estimates for workflow tasks incur in increased costs. If the estimated QoS is set too low, the workflow QoS might not be fulfilled, violating existing service level agreements. If it is set too high, service level agreements will be stricter than required, resulting in higher fees for external services and higher development costs for internal services.

In absence of other information, the user could derive initial values for these missing estimates according to a set of assumptions. For instance, the user might want to meet the workflow QoS by splitting the load equally over all tasks according to the probability and frequency they are invoked, requiring as little performance as possible. Nevertheless, it might be difficult to calculate these values by hand for complex compositions where some services are annotated.

In this work we propose two algorithms designed to assist the user in this process, following the assumptions above. The algorithms derive initial values for the response time under a certain load of all elements in a service composition model inspired on UML activity graphs. The algorithms have been successfully integrated into the models of an existing model-driven SOA methodology.

The structure of the rest of this text is as follows: in section 2 we describe the generic graph metamodel both inference algorithms work with. The algorithms are described in detail in sections 3 and 4. Section 5 discusses the adaptations required to integrate them into an existing model-driven SOA methodology. Section 6 evaluates the algorithms and the tools. Finally, related works are listed and some conclusions are offered, along with an outline of our future work.

2 Graph metamodel

The inference algorithms are defined for a generic graph metamodel. The metamodel is a simplification of UML activity diagrams, and uses mostly the same notation. The new global and local performance constraints are represented as stereotyped annotations.

A simplified UML class diagram for the ECore [7] graph metamodel is shown in figure 1. GRAPHs contain a set of FLOWNODEs and FLOWEDGEs and have a manual global PERFORMANCEANNOTATION, which indicates all paths in the graph should finish in less than *secsTimeLimit* seconds while handling *concurrentRequests* requests concurrently. There are several kinds of FLOWNODEs:

Activities encapsulate some behavior, described textually in the attribute *name*. Activities can have manual or automatic performance annotations.

Initial nodes are the starting execution points of the graphs. One per graph.

Final nodes end the current execution branch. There can be more than one.

Decision nodes select one execution branch among several, depending on whether the condition for their outgoing edge holds or not. Only the outgoing FlowEdges from a DecisionNode may have a non-empty *condition* and a *probability* less than 1.

Fork nodes split the current execution branch into several parallel branches.

Join nodes integrate several branches back into one, whether they started off from a decision node or a fork node. This is a simplification from UML, which uses different elements to integrate each kind of branch: join nodes and merge nodes, respectively.

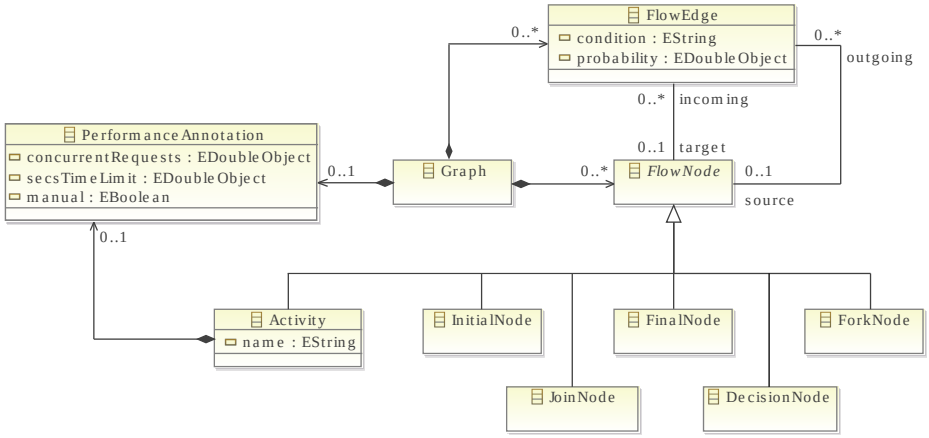

Fig. 1. Graph metamodel used in both algorithms

A sample model using this metamodel is shown in figure 2, which describes a process for handling an order. Execution starts from the initial node, and proceeds as follows:

1. The order is evaluated and is either accepted or rejected.
2. If rejected, close the order: we are done.
3. Otherwise, fork into 2 execution branches:
   (a) Create the shipping order and send it to the shipping partner.
   (b) Create the invoice, send it to the customer and receive its payment.
4. Once these 2 branches complete their execution, close the order.

There are two manual performance annotations: a global one (using the `<<gpc>>` stereotype) and a local one (using `<<pc>>`). The whole process must be able to serve 5 concurrent requests in less than 1 second, and evaluating an order should be done in 0.4 seconds at most with 5 concurrent requests.

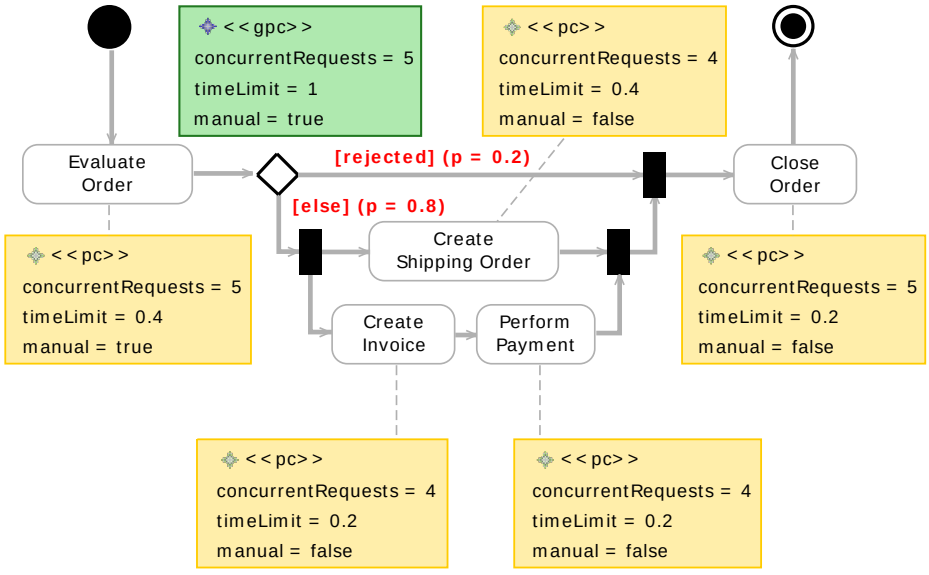

Fig. 2. Sample graph model

3 Inference of concurrent requests

The first algorithm infers the number of concurrent requests that must be handled at each activity in the graph before its time limit in order to meet the performance requirements of the whole graph. It performs a pre-order breadth-first traversal of every node and edge in the graph, starting from the initial node and caching intermediate results. The algorithm annotates each activity with its expected number of concurrent requests $C(x)$. The definition of $C(x)$ depends on the element type:

- InitialNode: $C(x)$ is the value of the attribute *concurrentRequests* of the global performance annotation of the graph.
- FlowEdge: $C(x) = P(x)C(s)$, where x is the edge, s is the source node of x, $P(x)$ is the value of the attribute *probability* of x and $C(s)$ is the computed number of concurrent requests for s. Most edges have $P(x) = 1$, except those which start from a decision node.
- JoinNode: the least common ancestor LCA(P) of all the parent nodes P is computed. This least common ancestor is the point from which the conditional or parallel branches started off. If LCA(P) is a decision node, $C(x) = \sum_{p \in P} C(p)$, as requests can only visit one of the branches. Otherwise, it is a fork node, and $C(x) = \max_{p \in P} C(p)$, as requests can visit all the branches at the same time.
  To compute the LCA of several nodes, the naive method described by Bender et al. [8] works best for the sparse graphs which usually result from modeling service compositions. For a graph with n vertexes and e edges, its

preprocessing step requires $O(n + e)$ operations and each query performed may require up to $O(n^2)$ operations. This only needs to be done once for each join node.
- Otherwise, $C(x) = C(i)$, where i is its only incoming edge. If the node is an activity with a manual performance annotation, the inferred value should match its value. Otherwise, the user will need to confirm if the annotation should be updated.

Using these formulas, computing C(Create Invoice) for the example shown in figure 2 would require walking back to the initial node, finding an edge with a probability of $p = 0.8$, no merge nodes and a global performance annotation for $G = 5$ concurrent requests. Therefore, C(Create Invoice) $= pG = 4$.

4 Inference of time limits

The algorithm required to infer the time limits for each activity in the graph is more complicated than the previous one. It adds time limits to the activities in each path from the initial node to any of the final nodes, ensuring the inferred time limits meet several assumptions.

This section starts with the requirements imposed on the results of the algorithm and lists the required definitions. The rest of the section describes the algorithm and shows an example of its application.

4.1 Design requirements

The inferred time limits must meet the following requirements:

1. The sum of the time limits of the activities in each path from the initial node to any of the final nodes must be less than or equal to the global time limit.
2. Available time should be split equally over the activities without any manual performance annotations. We cannot blindly assume that one activity will need to be more efficient than another. Ideally, we would like to split evenly the load over all activities.
3. Time limits should not be lower than strictly required: better performing systems tend to be more expensive to build and maintain.

4.2 Definitions

To describe the algorithm, several definitions are needed. These definitions consider paths as sets of activities, rather than sequences of nodes.

- $M(a)$ is true if the activity a has a manual time limit.
- $A(a)$ is true if the activity a has an automatic time limit.
- $U(p) = \{a | a \in p \wedge \neg(A(a) \vee M(a))\}$ is the set of all activities in the path p which do not have time limits. These are said to be *unrestricted*.

- $F(p) = \{a | a \in p \wedge \neg M(a)\}$ is the set of all activities in the path p which do not have *manual* time limits. These are said to be *free*.
- $T_L(G)$ is the manual time limit for every path in the graph G.
- $T_L(a)$ is the current time limit for the activity a.
- $T_L(p) = \sum_{a \in P \wedge (M(a) \vee A(a))} T_L(a)$ is the sum of the time limits in the path p.
- $T_M(p) = \sum_{a \in (p - F(p))} T_L(a)$ is defined as the sum of the *manual* time limits in the path p.
- $S_A(p, G) = T_L(G) - T_M(p)$ is the *slack* which can be split among the free activities in path p of the graph G.
- $T_E(p, G) = S_A(p, G)/(1 + |F(p)|)$ is an estimate of the automatic time limit each free activity in path p would obtain using only the information in that path. More restrictive paths will have lower values of $T_E(p, G)$.

4.3 Description of the algorithm

The algorithm follows these steps:

1. All automatic time limits are removed.
2. The set P of all paths from the initial node to each final node is calculated.
3. For each path $p \in P$ in ascending order of $T_E(p, G)$, so the more restrictive paths are visited first:
   (a) If $S_A(p, G) = 0$, the condition $|F(p)| > 0$ is evaluated. If it is true, there is at least one free activity in p for which no time limit can be set. The user is notified of the situation and execution is aborted. Otherwise, processing will continue with the next path.
   (b) If $S_A(p, G) < 0$, then $T_M(p) > T_L(G)$ holds, by definition. This means that the sum of the manual time limits in p exceeds the global time limit. The user is notified of the situation and execution is aborted.
   (c) Otherwise, $S_A(p, G) > 0$. If $|F(p)| > 0 \wedge |U(p)| > 0$, all activities in $U(p)$ will be updated to the time limit $(T_L(G) - T_L(p))/|U(p)|$.

4.4 Example

To infer the automatic time limits of the running example shown in figure 2, we first need to list all existing paths:

- $p_1 = \{\text{Evaluate Order}, \text{Close Order}\}$.
  $T_E(p_1, G) = (1 - 0.4)/(1 + 1) = 0.3$.
- $p_2 = \{\text{Evaluate Order}, \text{Create Shipping Order}, \text{Close Order}\}$.
  $T_E(p_2, G) = (1 - 0.4)/(1 + 2) = 0.2$.
- $p_3 = \{\text{Evaluate Order}, \text{Create Invoice}, \text{Receive Payment}, \text{Close Order}\}$.
  $T_E(p_3, G) = (1 - 0.4)/(1 + 3) = 0.15$.

p_3 has the lowest value for T_E, so it is visited first. "Create Invoice", "Perform Payment" and "Close Order" are in $U(p_3)$, so they are updated to $l = (1 - 0.4)/3 = 0.2$.

p_2 is visited next. Only "Create Shipping Order" is in $U(p_2)$, so it is updated to $l = (1 - 0.6)/1 = 0.4$.

p_1 is visited last. $U(p_1) = \emptyset$, so nothing is updated, and we are done.

5 Integration in a SOA methodology

The algorithms described above work on a generic graph metamodel which needs to be adapted to the metamodels used in specific technologies and methodologies. In this section we show how the metamodel and the algorithms were adapted and integrated into a SOA methodology: SODM+Testing.

Existing methodologies reduce the development cost of SOAs, but do not take testing aspects into account [9]. SODM+Testing is a new methodology that extends SODM [10] with testing tasks, techniques and models.

5.1 Metamodel adaptation

The original SODM service process and service composition metamodels were a close match to the generic graph metamodel described in section 2, as they were also based on UML activity graphs. Service process models described how a request for a service offered by the system should be handled, and service compositions provided details on how the tasks were split among the partners and what messages were exchanged between them.

SODM+Testing refactors these models using the generic graph metamodel as their core. Each class in the SODM process and composition metamodels can now be upcasted to the matching class of the generic graph metamodel both inference algorithms are based on. For instance, ProcessFlowEdge (class of all flow edges in the SODM service process metamodel) has become a subclass of FlowEdge (class of all flow edges in the generic graph metamodel).

The generic core differs slightly from that in figure 1. FlowEdges are split into ControlFlows and ObjectFlows, and performance annotations have also been split into several types according to their scope. In addition, the SODM+Testing service composition metamodel has been extended so activities can contain action sub-graphs with their own local and global performance annotations.

5.2 Tool implementation

SODM+T is a collection of Eclipse plug-ins that assists the SODM+Testing methodology with graphical editors for the SODM+Testing service process and composition models. It is available under the Eclipse Public License v1.0 [11].

To ensure the inference algorithms can be applied, models are validated automatically upon saving, showing the usual error and warning markers and Quick Fixes. Validation has been implemented using Epsilon Validation Language (EVL) scripts [12] and OCL constraints.

The algorithms are implemented in the Epsilon Object Language (EOL) and can be launched from the contextual menu of the graphical editor. The EOL scripts include two supporting algorithms for detecting cycles in graphs and computing the least common ancestor of two nodes [8].

6 Evaluation

In this section we will evaluate each algorithm, pointing out their current limitations and how we plan to overcome them. Later on, we will evaluate their theoretical performance. Finally, we will discuss the current state of SODM+T.

6.1 Inference of concurrent requests

This algorithm performs a breadth-first traversal on the graph, and assumes the number of concurrent requests has already been computed for all parent nodes. For this reason, it is currently limited to acyclic graphs. We intend to extend the model to allow loops, limiting their maximum expected number of iterations.

The algorithm requires edges to be manually annotated with probabilities. These could be simply initialized to assume each branch in each decision node has the same probability of being activated, in absence of other information.

The algorithm assumes only the current workflow is being run, thereby providing a best-case scenario. More realistic estimations would be obtained if restrictions from all workflows in the system were aggregated.

Assuming the naive LCA algorithm has been used, this algorithm completes its execution for a graph with n nodes after $O(n^3)$ operations, as its outer loop visits each node in the graph to perform $O(n^2)$ operations for JoinNodes and $O(1)$ operations for the rest. This is a conservative upper bound, as the $O(n^2)$ time for the naive LCA algorithm is for dense graphs, which are uncommon in service compositions.

6.2 Inference of time limits

As there must be a finite number of paths in the graph, the current version requires the graph to be acyclic, just as the previous algorithm. The algorithm could reduce paths with loops to regular sequences, where the activities in the loop would be weighted to take into account the maximum expected number of iterations.

The algorithm only estimates maximum deterministic time limits. Probability distributions are not computed, as it does not matter which path is actually executed: all paths should finish in the required time.

In addition, the user cannot adjust the way in which the available slack is distributed to each task. For instance, the user might know that a certain task would usually take twice as much time as some other task. Weighting the activities as suggested above would also solve this problem.

Finally, like the previous algorithm, the time limit inference algorithm is currently limited to the best-case scenario where only the current workflow is being run.

A formal proof of the correctness of the time limit inference algorithm is outside the scope of this paper. Nevertheless, we can show that the results produced by the algorithm meet the three design requirements listed in § 4.1. We assume

the model is valid (acyclic and weakly connected, among other restrictions) and that the algorithm completes its execution successfully:

1. The first requirement is met by sorting the paths from most to least restrictive. For the i-th path p_i, only the nodes for which p_i provides the strictest constraints (that is, $U(p_i)$) are updated so $T_L(p_i) < T_L(G)$.
   Since the activities from the previous (and stricter) paths were not changed, $T_L(p_j) < T_L(G), j \leq i$. Once all paths have been processed, $T_L(p) < T_L(G)$ holds for all $p \in P$.
2. The second requirement is met by simply distributing equally the available slack using a regular division.
3. The third requirement is met by visiting the most restrictive paths first, so activities obtain the strictest time limits as soon as possible, leaving more slack for those who can use it.

The running time of the algorithm is highly dependent on the number of unique paths in the graph. With no loss of generality, we assume all forks and decision nodes are binary. If there are b decision and fork nodes in total among the n nodes, there will be 2^b unique paths to be considered.

Enumerating the existing paths requires a depth-first traversal of the graph, which may have to consider the $O(n^2)$ edges in the graph. Each path in the graph will have to be traversed a constant number of times to compute $T_E(p, G)$, $S_A(p, G)$, $|F(p)|$ and other formulas which require $O(n)$ operations.

Therefore, the algorithm requires $O(n^2 + n2^b)$ time. We can see that the number of forks and decisions has a large impact on the running time of the algorithm. This could be alleviated by culling paths subsumed in other paths. We are also looking into alternative formulations which do not require all paths to be enumerated.

6.3 SODM+T

SODM+T has been successfully used to model the abstract logic required to handle a business process from a medium-sized company, generating performance annotations for graphs with over 60 actions nested in several activities.

However, SODM+T could be improved: future versions will address some of the issues found during this first experience. WSDL descriptions and test cases (for tools such as soapUI [13] or JUnitPerf [14]) need to be generated from the models. Performance annotations should be more fine-grained, letting the user specify only one of the attributes so the other is automatically inferred.

Finally, the tools could be adapted to other metamodels more amenable to code generation. Business Process Modeling Notation [3] models are a good candidate, as partial automated translations to executable forms already exist [15].

7 Related work

Several UML profiles for modeling the expected performance of a system have been approved by the Object Management Group: SPTP [16], QFTP [17] and

MARTE [18]. To study the feasibility of our approach, we have kept our constraints much simpler, but we intend to adapt them to either SPTP or QFTP in the future.

Existing works on performance estimation in workflows focus on aggregating workflow QoS from task QoS, unlike our approach, which estimates task QoS from workflow QoS. Silver et al. [19] annotated each task in a workflow with probability distributions for their running times, and collected data from 100 simulation runs to estimate the probability distribution of the running time of the workflow and validate it using a Kolmogorov-Smirnov goodness-of-fit test. Simulating the workflow allows for flexibly modeling the stochastic behavior of each service, but it has a high computational overhead due to the number of simulation runs that are required. Our work operates directly on the graph, imposing less computational overhead, but only modeling deterministic maximum values.

Other authors convert workflows to an existing formalism and solve it using an external tool. The most common formalisms are layered queuing networks [20], stochastic Petri networks [21] and process algebra specifications [22]. These formalisms are backed by in-depth research and the last two have solid mathematical foundations in Markov chain theory. However, they introduce an additional layer of complexity which might discourage some users from applying these techniques. Our work operates on the models as-is, with no need for conversions. This makes them easier to understand.

Finally, some authors operate directly on the workflow models, as our approach does. The SWR algorithm proposed by Cardoso et al. [4] computes workflow QoS by iteratively reducing its graph model to a single task. SWR only works with deterministic values, like our algorithms, but its QoS model is more detailed than ours, describing the cost, reliability and minimum, average and maximum times for each service. We argue that though average and minimum times might be useful for obtaining detailed estimations, they provide little value for top-down development, which is only interested in establishing a set of constraints which can be used for conducting load testing and monitoring for each service. However, cost and reliability could be interesting in this context as well.

It is interesting to note that Cardoso et al. combine several data sources to estimate the QoS of each task [4]: designer experience, previous times in the current instance, previous times in all instances of this workflow, and previous times in all workflows. The designer is responsible for specifying the relative importance of each data source. Results from our algorithms could be used as yet another data source.

Our metamodels have been integrated into SODM+Testing, a SOA model-driven methodology which extends the SODM methodology [10] with testing aspects. Many other SOA methodologies exist: for instance, Erl proposes a high-level methodology focused on governance aspects in his book [1] and IBM has defined the comprehensive (but proprietary) SOMA methodology [23]. SODM was selected as it is model-driven and strikes a balance between scope and cost.

8 Conclusions and future work

Service compositions allow several WS to be integrated as a single WS which can be flexibly reconfigured as business needs change. However, setting initial performance requirements for the integrated services in a top-down methodology so the entire composition works as expected without incurring additional costs is difficult. At an early stage of development, there is not enough information to use more comprehensive techniques.

This work has presented the first version of two algorithms which infer missing information about response times under certain loads from models inspired in UML activity graphs. Though a formal proof was outside the scope of this paper, we have found them to meet their design requirements: they infer local constraints which meet the global constraints, distribute the load equally over the composition and allow for as much slack as possible.

These algorithms and their underlying metamodel have been successfully integrated into a model-driven SOA methodology, continuing the previous work shown in [9]. The tools [11] implement a graphical editor which allows the user to define performance requirements at several levels and offers automated validation and transformation.

Nevertheless, the algorithms and tools could be improved. The underlying metamodel should allow for partially automatic annotations and use weights for distributing the available slack between the activities. In the future, the annotations could be adapted to the OMG SPTP [16], QFTP [17] or MARTE [18] UML profiles.

The algorithms will be revised to handle cyclic graphs and aggregate constraints for a service from several models. The performance of the time limit inference algorithm depends on the number of unique graphs from the initial node to each final node in the tree. We are looking into ways to reduce the number of paths which need to be checked.

Finally, it is planned to adapt the tools to service composition models more amenable to executable code generation, such as BPMN [3], and to generate test cases for existing testing tools such as soapUI [13] or JUnitPerf [14].

References

1. Erl, T.: SOA: Principles of Service Design. Prentice Hall, Indiana, EEUU (2008)
2. OASIS: Web Service Business Process Execution Language (WS-BPEL) 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html` (April 2007)
3. Object Management Group: Business Process Modeling Notation (BPMN) 1.2. `http://www.omg.org/spec/BPMN/1.2/` (January 2009)
4. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web **1**(3) (April 2004) 281–308
5. Hwang, S., Wang, H., Tang, J., Srivastava, J.: A probabilistic approach to modeling and estimating the QoS of web-services-based workflows. Information Sciences **177**(23) (2007) 5484–5503

6. Yu, T., Zhang, Y., Lin, K.: Efficient algorithms for web services selection with end-to-end QoS constraints. ACM Transactions on the Web **1**(1) (2007)
7. Eclipse Foundation: Eclipse Modeling Framework. `http://eclipse.org/modeling/emf/` (2010)
8. Bender, M.A., Pemmasani, G., Skiena, S., Sumazin, P.: Finding least common ancestors in directed acyclic graphs. Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01) (2001) 845–853
9. García-Domínguez, A., Medina-Bulo, I., Marcos-Bárcena, M.: Hacia la integración de técnicas de pruebas en metodologías dirigidas por modelos para SOA. In: Actas de las V Jornadas Científico-Técnicas en Servicios Web y SOA, Madrid, España (October 2009)
10. de Castro, M.V.: Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de servicios web. PhD thesis, Universidad Rey Juan Carlos (March 2007)
11. García-Domínguez, A.: Homepage of the SODM+T project. `https://neptuno.uca.es/redmine/projects/sodmt` (March 2010)
12. Kolovos, D., Paige, R., Rose, L., Polack, F.: The Epsilon Book. `http://www.eclipse.org/gmt/epsilon` (2010)
13. eviware.com: Homepage of soapUI. `http://www.soapui.org/` (2009)
14. Clark, M.: JUnitPerf. `http://clarkware.com/software/JUnitPerf.html` (October 2009)
15. Küster, T., Heßler, A.: Towards transformations from BPMN to heterogeneous systems. In Mecella, M., Yang, J., eds.: BPM2008 Workshop Proceedings, Milan, Italy (September 2008)
16. Object Management Group: UML Profile for Schedulability, Performance, and Time (SPTP) 1.1. `http://www.omg.org/spec/SPTP/1.1/` (January 2002)
17. Object Management Group: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QFTP) 1.1. `http://www.omg.org/spec/QFTP/1.1/` (April 2008)
18. Object Management Group: UML Profile for Modeling and Analysis of Real-Time Embedded systems (MARTE) 1.0. `http://www.omg.org/spec/MARTE/1.0/` (November 2009)
19. Silver, G.A., Maduko, A., Rabia, J., Miller, J., Sheth, A.: Modeling and simulation of quality of service for composite web services. In: Proceedings of 7th World Multiconference on Systemics, Cybernetics and Informatics, International Institute of Informatics and Systems (November 2003)
20. Petriu, D.C., Shen, H.: Applying the UML Performance Profile: Graph Grammar-based Derivation of LQN Models from UML Specifications. In: Proceedings of the 12th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools (TOOLS 2002). Volume 2324 of Lecture Notes in Computer Science. Springer Berlin, London, UK (2002) 159—177
21. López-Grao, J.P., Merseguer, J., Campos, J.: From UML activity diagrams to Stochastic Petri nets: application to software performance engineering. SIGSOFT Softw. Eng. Notes **29**(1) (2004) 25–36
22. Tribastone, M., Gilmore, S.: Automatic extraction of PEPA performance models from UML activity diagrams annotated with the MARTE profile. In: Proceedings of the 7th International Workshop on Software and Performance, Princeton, NJ, USA, ACM (2008) 67–78
23. Ghosh, S., Arsanjani, A., Allam, A.: SOMA: a method for developing service-oriented solutions. IBM Systems Journal **47**(3) (2008) 377–396