Supporting Non-Functional Requirements in Services Software Development Process: an MDD Approach

Valeria de Castro[1], Martin A. Musicante[2], Umberto Souza da Costa[2], Plácido A. de Souza Neto[3], and Genoveva Vargas-Solar[4]

[1] Universidad Rey Juan Carlos – Móstoles, Spain
Valeria.deCastro@urjc.es
[2] Federal University of Rio Grande do Norte (UFRN) – Natal-RN, Brazil
{mam,umberto}@dimap.ufrn.br
[3] Federal Technological Institute of Rio Grande do Norte (IFRN) – Natal-RN, Brazil
placido.neto@ifrn.edu.br
[4] French Council of Scientific Research (CNRS) – Grenoble, France
Genoveva.Vargas-Solar@imag.fr

Abstract. This paper presents the πSOD-M method, an extension to the Service-Oriented Development Method (SOD-M) to support the development of services software by considering their functional and non-functional requirements. Specifically, πSOD-M proposes: *(i)* meta-models for representing non-functional requirements at different abstraction levels; *(ii)* model-to-model transformation rules, useful to semi-automatically refine Platform Independent Models into Platform Specific Models; and *(iii)* rules to transform Platform Specific Models into concrete implementations. In order to illustrate our proposal, the paper also describes how to apply the methodology to develop a proof of concept.

Keywords: MDD, Service Oriented Applications, Non-functional Properties

1 Introduction

Model Driven Development (MDD) [12] is a top-down approach proposed by the Object Management Group (OMG)[5] for designing and developing software systems. MDD provides a set of guidelines for structuring specifications by using *models* to specify software systems at different levels of abstraction or *viewpoints*:

- *Computation Independent Models (CIM)*: this viewpoint represents the software system at its highest level of abstraction. It focusses on the system environment and on business and requirement specifications. At this level, the structure of the system and its processing details are still unknown or undetermined.
- *Platform Independent Models (PIM)*: this viewpoint focusses on the system functionality, hiding the details of any particular platform.

[5] http://www.omg.org/mda.

- *Platform Specific Models (PSM)*: this viewpoint focusses on the functionality, in the context of a particular implementation platform. Models at this level combine the platform-independent view with specific aspects of the target platform in order to implement the system.

Besides the notion of models at each level of abstraction, MDD requires *model transformations* between levels. These transformations may be automatic or semi-automatic and implement the refinement process between levels.

MDD has been applied for developing service-oriented applications. In Service-Oriented Computing [20], pre-existing services are combined to produce applications and provide the business logic. The selection of services is usually guided by the functional requirements of the application being developed. Some methodologies and techniques have been proposed to help the software developer in the specification of functional requirements of the business logic, such as the Service-Oriented Development Method (SOD-M) [9].

Ideally, non-functional requirements such as security, reliability, and efficiency would be considered along with all the stages of the software development. The adoption of non-functional specifications from the early states of development can help the developer to produce applications that are capable of dealing with the application context. Non-functional properties of service-oriented applications have been addressed in academic works and standards. Dealing with these kind of properties involves the use of specific technologies at different layers of the SOC architecture, for instance during the description of service APIs (such as WSDL[8] or REST [13]) or to express service coordinations (like WS-BPEL [1]).

Protocols and models implementing non-functional properties assume the existence of a global control of the artefacts implementing the application. They also assume that each service exports its interface. So, the challenge of supporting non-functional properties is related to *(i)* the specification of the business rules of the application; and *(ii)* dealing with the technical characteristics of the infrastructure where the application is going to be executed.

This paper presents πSOD-M, a methodology for supporting the development of service-oriented applications by taking into account both functional and non-functional requirements. This methodology aims to: *(i)* improve the development process by providing an abstract view of the application and ensuring its conformance to the business requirements; *(ii)* reduce the programming effort through the semi-automatic generation of models for the application, to produce concrete implementations from high-abstraction models. Accordingly, the remainder of the paper is organized as follows: Section 2 gives an overview of the SOD-M approach; Section 3 presents πSOD-M, the methodology that we propose to extend SOD-M; Section 4 shows the development of a proof of concept; Section 5 describes related works; and Section 6 concludes the paper.

2 SOD-M

The Service-Oriented Development Method (SOD-M) [9] adopts the MDD approach to build service-based applications. SOD-M considers two points of view:

(i) business, focusing on the characteristics and requirements of the organization; and *(ii) system requirements*, focusing on features and processes to be implemented in order to build service-based applications in accordance to the business requirements. In this way, SOD-M simplifies the design of service-oriented applications, as well as their implementation using current technologies.

SOD-M provides a framework with models and standards to express functionalities of applications at a high-level of abstraction. The SOD-M meta-models are organized into three levels: CIM (*Computational Independent Models*), PIM (*Platform Independent Models*) and PSM (*Platform Specific Models*). Two models are defined at the CIM level: *value model* and *BPMN model*. The PIM level models the entire structure of the application, as the PSM level provides transformations towards more specific platforms. The PIM-level models are: *use case*, *extended use case*, *service process* and *service composition*. The PSM-level models are: *web service interface*, *extended composition service* and *business logic*.

At the CIM level, the *value model* describes a business scenario as a set of values and activities shared by business actors. The *BPMN model* describes a business process and the corresponding environment. At the PIM level, the *use case model* represents a business service, as the *extended use case model* contains behavioral descriptions of features to be implemented. The *service process model* describes a set of activities to be performed in order to implement a business service. Finally, the *service composition model* represents the complete flow of a business system. This model is an extension of the service process model.

The SOD-M approach includes transformations between models: *CIM-to-PIM, PIM-to-PIM* and *PIM-to-PSM* transformations. Given an abstract model at the CIM level, it is possible to apply transformations for generating a model of the PSM level. In this context, it is necessary to follow the process activities described by the methodology. These three SOD-M levels have no support for describing non-functional requirements. The following section introduces πSOD-M, the extension that we propose for supporting these requirements.

3 πSOD-M

πSOD-M provides an environment for building service compositions by considering their non-functional requirements. πSOD-M proposes the generation of a set of models at different abstraction levels, as well as transformations between these models. πSOD-M includes non-functional specifications through four meta-models that extend the SOD-M meta-models at the PIM and PSM levels (see Figure 1): π-*UseCase*, π-*ServiceProcess*, π-*ServiceComposition* and π-*PEWS*.

The π-*UseCase* meta-model describes functional and non-functional requirements. Non-functional requirements are defined as *constraints* over processing steps and data. The π-*ServiceProcess* meta-model defines the concept of *service contract* to represent restrictions over data and actions that must be performed upon certain conditions. The π-*ServiceProcess* meta-model gathers the constraints described in the π-*UseCase* model into contracts that are associated with services. The π-*ServiceComposition* meta-model provides the concept

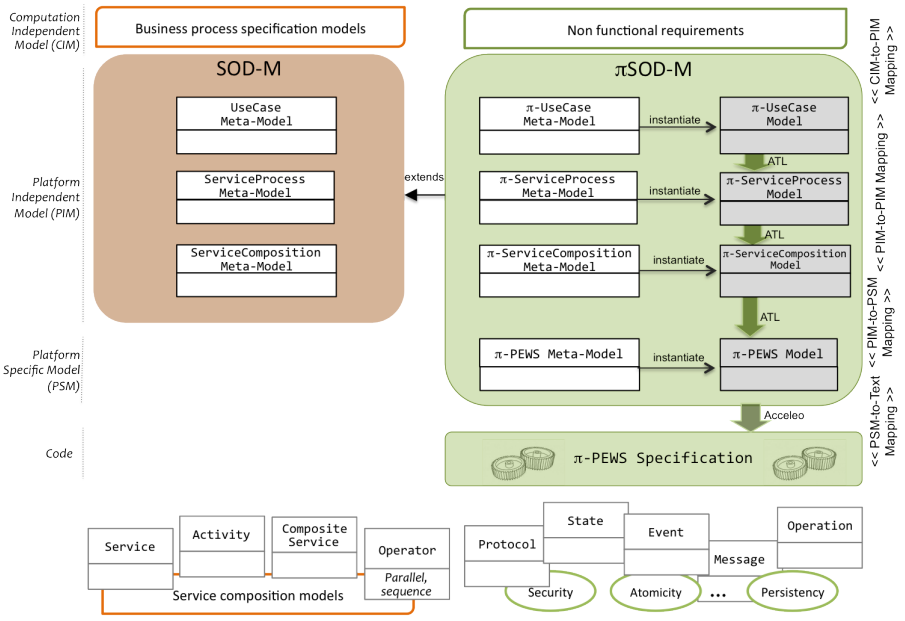


Fig. 1: πSOD-M.

of *Policy* [11] which put together contracts with similar non-functional requirements. For instance, security and privacy restrictions may be grouped into a security policy. π-*ServiceComposition* models can be refined into PSMs. Policies are associated to service operations and combine *constraints* and *reactive recovery actions*. Constraints are restrictions that must be verified during the execution of the application. Failure to verify some constraint will trigger an exceptional behavior to execute the corresponding recovery action. An example of policy is the requirement of authentication for executing some of the system functions. The action associated to this policy may perform the authentication of the user. At the PSM level we have lower-level models that can be automatically translated into actual computer programs. The π-*PEWS* meta-model is the PSM adopted in this work (see Figure 1). π-*PEWS* models are textual descriptions of service compositions that can be translated into PEWS [3] code.

Thus, πSOD-M proposes a development process based on the definition of models (instances of the meta-models) and transformations between models. There are two kinds of transformations: model-to-model transformations, used to refine the specification during the software process; and model-to-text transformations, used to generate code in the last step of the software process. Notice that other composition languages, such as BPEL [1], can be used as target of the proposed software process. Another target language can be supported by defining the corresponding PIM-to-PSM and PSM-to-text transformations.

πSOD-M environment is built on the top of Eclipse. We also used the Eclipse Modelling Framework (EMF) to define, edit and handle (meta)-models. To automate the transformation models we use ATL [15] and Acceleo.

In the next section we develop an example, in order to illustrate our proposal. The example will show the actual notation used for models.

4 Proof of Concept: *Tracking Crimes*

Consider a tracking crime application where civilians and police share information about criminality in given zones of a city. Civilian users signal crimes using Twitter. Police officers can notify crimes, as well as update information about solving cases. Some of these information are confidential while other can be shared to the community of users using this application. Users can track crimes in given zones. Crime information stored by the system may be visualized on a map. Some users have different access rights than others. For example, police officers have more access rights than civilians.

In order to provide these functionalities, the application uses pre-existing services to provide, store and visualize information. The business process defines the logic of the application and is specified in terms of tasks. Tasks can be performed by people or computers.

The business process and requirements specifications presented in Figure 2 are instances of the Computation-Independent models of Figure 1. The business process is represented as a graph while requirements are given as text boxes.

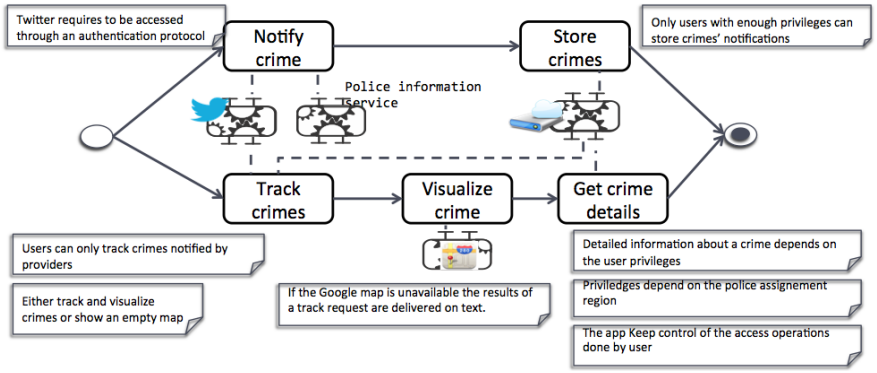


Fig. 2: Business process for the tracking crime example.

In our example, crime processing can start with one of two tasks: *(i) notify a crime*, or *(ii) track a crime*. Notified crimes are stored in a database. Tracked crimes are visualized on a map. The used can ask for detailed information. The application is built upon four services: Twitter and an ad-hoc police service, for

notifying crimes; Amazon, used as persistence service; and Google Maps, for locating and displaying crimes on a map.

Non-functional requirements are specified by rules and conditions to be verified during the execution of tasks. In our example we have the following non-functional requirements:

- Twitter requires authentication and allows three login failures before blocking.
- Crime notification needs privileged access.
- Civilian users can only track crimes for which they have clearance: civilian population cannot track all the crimes notified by the police.
- If Google Maps is unavailable, the results are delivered as text.
- Querying about crimes without having proper clearance yields an empty map.
- Access rights to detailed information depends on user clearance and zone assignment for police officers.
- The application maintains a detailed log.

The idea about these requirements is to leave the application logic expressed by functional requirements as independent as possible from exceptional situations (like the unavailability of a service) and from conditions under which services are called (for instance, through an authentication protocol). These requirements can be weaved as activities and branches of the composition or implemented apart. The second option is better because it makes the maintenance and evolution of applications easier. For instance, the services called by the application are not hard coded (Twitter and Google Maps in the example), neither the actions to deal with exceptions (replacing another map service or doing nothing).

All the system restrictions are modelled as constraints in this example. πSOD-M provides three types of constraints: *value*, *business* and *exceptions behaviour* constraints. Each use case (model) can be associated to one or more constraints[6].

π-UseCase model: our example has five use cases which represent the functions (tasks) and constraints of the system (Figure 3). We do not detail the functional part of the specification, due to lack of space. The constraints defined are:

- The Notify crime task requires that the user is logged in. This is an example of a *value constraint*, where the value associated to the condition depends on the semantics of the application. In this case, it represents the maximum number of allowed login attempts;
- The Store crimes task requires the verification of the user's clearance (also a value constraint).
- In order to perform the Track crimes task, the contact list of the requesting user must include the user that notified the crime. This is an example of *business constraint*. Additionally the requesting user must be logged in.

[6] For a more comprehensive account of πSOD-M the reader can refer to [23].

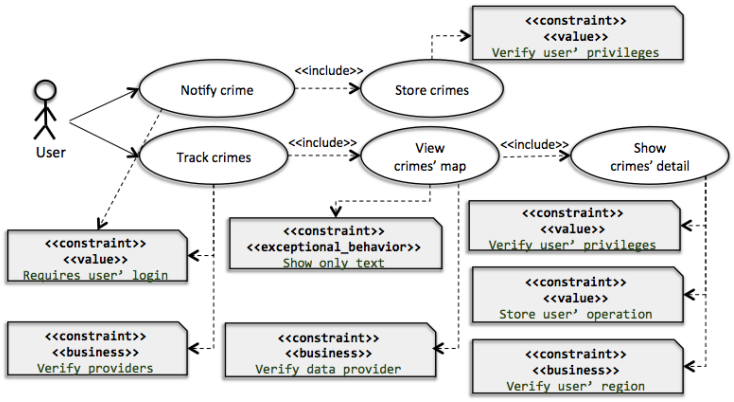


Fig. 3: π-UseCase Model.

- For the View crimes' map task, the specification defines that if the service Google Maps is not available, the result is presented as text. This is an example of *exceptional behaviour constraint*. The availability of the Google Maps service is verified by a *business constraint*.
- The Show crimes' details task is specified to have three constraints: A *value constraint* is defined to verify the user's clearance level; a *business constraint* is used to ensure that the user's clearance is valid for the geographic zone of the crime; another *value constraint* defines that the log is to be maintained.

A model-to-model transformation is defined to refine the π-*UseCase* model into the corresponding π-*ServiceProcess* model, a more detailed representation. This (semi-automatic) transformation is supported by a tool (described in [23]).

π-*ServiceProcess model:* in this model task nodes of the π-UseCase model are better detailed, by refining the control and data flows, and its constraints are transformed into *contracts* (pre- and post-conditions). The π-ServiceProcess model describes the activities of the application and defines contracts for each activity or parts of the application (Figure 4). The main transformations are:

- Tasks of the previous model are transformed into *actions*;
- Actions are grouped into *activities* (in accordance to the business logic).
- Constraints of the π-*UseCase* model are transformed into assertions.

π-*ServiceComposition model:* this model refines the previous model by using the activities to produce the workflow of the application. The model serves to identify those entities that collaborate with the service process by providing services to execute actions. This model identifies the services and functions that correspond to each action in the business process.

In the case of our crime tracking example, the model produced from the π-*ServiceProcess* model of Figure 4 is given in Figures 5a and 5b. Figure 5a shows

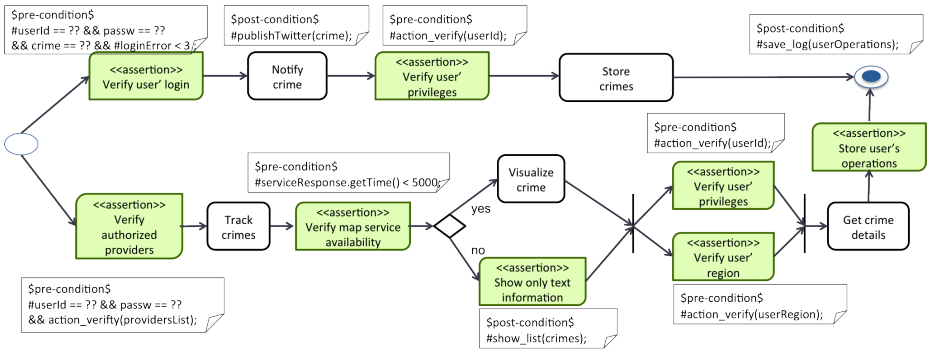


Fig. 4: π-*ServiceProcess* Model.

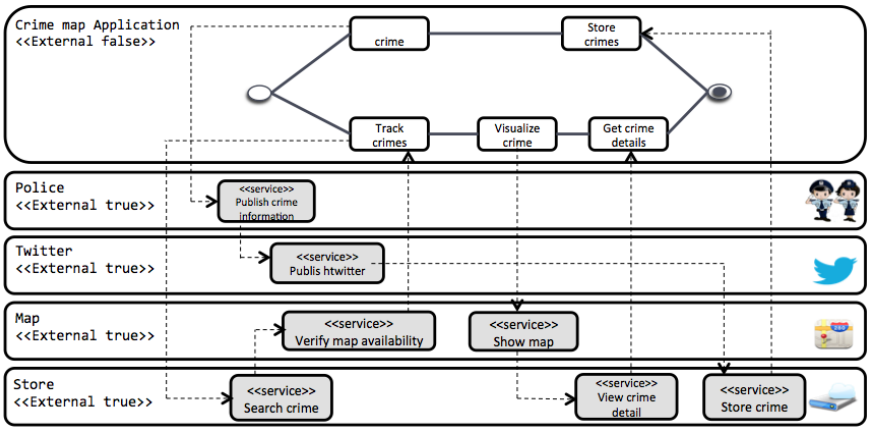
how the crime application interacts with its *business collaborators* (external services and entities). The interaction occurs by means of function calls (denoted by dotted lines in the figure). Figure 5b shows the definition of three *policies*, which define rules for service execution. In our case we have policies for *Security*, *Performance* and *Persistence*.

π-*PEWS Model:* this model is produced by a model-to-text transformation that takes a π-*ServiceComposition* model and generates π-*PEWS* specification code. This code is a service composition program that can be compiled into an executable composition. π-*PEWS* models are expressed in a variant of the PEWS composition language. The π-PEWS program generated from the model in Figure 5 is partially presented in Figure 6. The figure shows a simplified program code, produced in accordance to the following guidelines:

- Namespaces, identifying the addresses of external services are produced from the Business Collaborators of the higher-level model. We define four of them, corresponding to the Police, Twitter, Google Map and Amazon partners.
- Specific operations exported by each business collaborator are identified as an *operation* of the program (to each operation is given an `alias`).
- The workflow in Figure 5a is translated into the text at line 11.
- *Contracts* are defined in π-PEWS as having pre-conditions (`requires`), post-conditions (`ensures`) and actions (`OnFailureDo`) to be executed whenever a condition is not verified. Contracts are generated from Policies (Figure 5b).

5 Related work

Over the last years, a number of approaches have been proposed for the development of web services. These approaches range from the proposal of new



(a) π-ServiceComposition Model.

```
<<policy>>
$securityPolicy$
%security%

Rule R1{
@PRE@
#userId == ?? && passw == ??
&& crime == ?? && #loginError < 3#
!call(publishTwitter(crime))!
} Rule R2{
@PRE@
#userId == ?? && passw == ??
&& action_verifty(providersList)#
!call( trackCrimes())!
} Rule R3{
@PRE@
#action_verify(userId) &&
action_verify(userRegion);#
!call(getCrimeDetail())!
}
```

```
<<policy>>
$performancePolicy$
%performance, availability%

Rule R1{
@PRE@
#serviceResponse.getTime()
 < 5000;#
!call(showMap());!
}
Rule R2{
@POST@
#NOT(serviceResponse.getTi
me() < 5000);#
!show_list(crimes);!
```

```
<<policy>>
$percistencePolicy$
%percistence%

Rule R1{
@PRE@
#action_verify(userId);#
!call(storeCrime(crime))!
} Rule R2{
@POST@
#storeCrime().result == true &&
 getCrimeDetail.result == true#
!save_log(userOperations)!
}
```

(b) π-ServiceComposition Policies.

Fig. 5: Service Composition and Policies.

languages for web service descriptions [1, 21] to techniques to support phases of the development cycle of this kind of software [6]. In general, these approaches concentrate on specific problems, like supporting transactions or QoS, in order to improve the security and reliability of service-based applications. Some proposals address service composition: workflow definition [25, 18] or semantic equivalence between services [3].

Works dealing with non-functional properties in service-oriented development can be organized in two main groups: those working on the *modelling of particular non-functional properties or QoS attributes* and those proposing *architectures or frameworks to manage and validate QoS attributes in web service composition processes*. The first group considers specific non-functional concerns (e.g., security) which is modelled and then associated to functional models of the application. The work of Chollet et al. [7] defines a proposal to associate

```
//Namespaces specify service URI
1 namespace twitter = www.twitter.com/service.wsdl
2 namespace googlemaps = maps.googleapis.com/maps/api/service
3 namespace amazondynamodb = rds.amazonaws.com/doc/2010-07-28/AmazonRDSv4.wsdl
4 namespace police = www.police.fr/service.wsdl
//Operations
5 alias publishTwitter = portType/publishTwitter in twitter
6 alias searchCrime = portType/searchCrime in amazondynamodb
7 alias showMap = portType/showMap in googlemaps
//Services
8  service notifyCrime = publishCrime  . publishTwitter
9  service trackCrime= searchCrime . verifyService
10 Service visualizeCrime =  showMap . getCrimeDetail
//Path
11 (notifyCrime.storeiCrime) || (trackCrime.visualizeCrime.getCrimeDetail)
//Contracts
12 defContract notifyCrimeContract{  isAppliedTo: notifyCrime
13   requires: userId == ?? && passw == ?? && req(notifyCrime)  < 3
14        (OnFailureDo: NOT(action_publish(crime));
15   ensures: publishTwitter(crime) == true  (OnFailureDo: skip); }
```

Fig. 6: π-PEWS code for the crime tracking example (partial, simplified).

non-functional quality properties (security properties) to functional activities in a web service composition model. Schmeling et al. [22] present an approach and also a toolset for specifying and implementing non-functional concerns in web service compositions. Non-functional concerns are modelled and then related to a service composition represented in a BPMN diagram. Ovaska et al. [19] present an approach to support quality management at design time. Quality requirements are modelled in a first phase and then represented in an architectural model where quality requirements are associated to some components of the model. We propose to model functional and non-functional properties at the same time during the software process. We claim what this approach simplifies the web service development task. Non-functional properties are represented in our work as constraints and policies but more general quality model such as the proposed by [14, 17] could be taken in consideration in further works.

The second group of works dealing with non-functional requirements for services propose specific architectures or frameworks to manage and validate QoS attributes in service composition processes [26, 4, 16]. Our proposal is similar to these, but focusses on more general non-functional properties.

Despite the variety of techniques proposed, there is not yet a consensus on a software methodology for web services. Some methodologies address the service-based development towards a standard or a new way to develop reliable applications. SOD-M and SOMF [5] are MDD approaches for web services; S-Cube [20] is focused on the representation of business processes and service-based development; SOMA [2] is a methodology for SOA solutions; DEVISE [10] is a methodol-

ogy for building service-based infrastructure for collaborative enterprises. Other proposals include the WIED model [24], that acts as a bridge between business modelling and design models on one hand, and traditional approaches for software engineering applied to SOC on the other hand.

6 Conclusions

This paper presented the πSOD-M software method for specifying and designing service-based applications in the presence of non-functional constraints. Our proposal enhances the SOD-M method with constraints, policies and contracts to consider non-functional constraints of applications. We implemented the proposed meta-models on the Eclipse platform and we illustrated the approach by developing a simple application.

πSOD-M is being used in an academic environment. So far, the preliminary results indicate that πSOD-M approach is useful for the development of complex web service applications. We are now working on the definition of a PSM-level meta-model to generate BPEL programs (instead of π-PEWS) in order to be able to generate code for the de-facto service coordination standard.

Acknowledgements

This research is partly supported by the National Institute of Science and Technology for Software Engineering (INES[7]), funded by CNPq (Brazil), grants 573964/2008-4 and 305619/2012-8; CAPES/UdelaR (Brazil/Uruguay) grant 021/ 2010; CAPES/STIC-AmSud (Brazil) grant 020/2010); MASAI project (TIN-2011-22617) financed by the Spanish Ministry of Science and Innovation and the Spanish Network on Service Science (TIN2011-15497-E) financed by the Spanish Ministry of Competitiveness and Economy.

References

1. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weeranwarana, S.: Business Process Execution Language for Web Services. Available at http://www-128.ibm.com/developerworks/library/specification/ws-bpel/ (2003)
2. Arsanjani, A.: SOMA: Service-Oriented Modeling and Architecture. Technical report, IBM, http://www.ibm.com/developerworks/library/ws-soa-design1 (2004)
3. Ba, C., Halfeld-Ferrari, M., Musicante, M.A.: Composing Web Services with PEWS: A Trace-Theoretical Approach. In: ECOWS 2006. (2006) 65–74
4. Babamir, S.M., Karimi, S., Shishechi, M.R.: A Broker-Based Architecture for Quality-Driven Web Services Composition. In: Proc. CiSE 2010. (2010)
5. Bell, M.: Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture. John Wiley (2008)
6. Börger, E., Cisternino, A., eds.: Advances in Software Engineering (Revised Tutorial Lectures). In Börger, E., Cisternino, A., eds.: Lipari Summer School. Volume 5316 of LNCS., Springer (2008)

[7] www.ines.org.br

7. Chollet, S., Lalanda, P.: An Extensible Abstract Service Orchestration Framework. In: Proc. ICWS 2009, IEEE (2009) 831–838
8. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. Technical report, World Wide Web Consortium (2001) Available at http://www.w3.org/TR/wsdl.
9. de Castro, V., Marcos, E., Wieringa, R.: Towards a service-oriented MDA-based approach to the alignment of business processes with IT systems: From the business model to a web service composition model. IJCIS **18**(2) (2009)
10. Dhyanesh, N., Vineel, G.C., Raghavan, S.V.: DEVISE: A Methodology for Building Web Services Based Infrastructure for Collaborative Enterprises. In: Proc. WETICE'03, USA, IEEE Computer Society (2003)
11. Espinosa-Oviedo, J.A., Vargas-Solar, G., Zechinelli-Martini, J.L., Collet, C.: Policy driven services coordination for building social networks based applications. In: Proc. of SCC'11, Work-in-Progress Track, USA, IEEE (2011)
12. Favre, L.: A Rigorous Framework for Model Driven Development. In: Advanced Topics in Database Research, Vol. 5. Chapter I, IGP. (2006) 1–27
13. Fielding, R.T.: REST: Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine (2000)
14. Goeb, A., Lochmann, K.: A software quality model for soa. In: Proc. WoSQ '11, ACM (2011) 18–25
15. Group, A.: ATL: Atlas Transformation Language. Technical report, ATLAS Group, LINA & INRIA (February, 2006)
16. Karunamurthy, R., Khendek, F., Glitho, R.H.: A novel architecture for Web service composition. J. of Network and Computer Applications **35**(2) (2012) 787 – 802
17. Klas, M., Heidrich, J., Munch, J., Trendowicz, A.: Cqml scheme: A classification scheme for comprehensive quality model landscapes. In: SEAA '09. (2009) 243–250
18. Musicante, M.A., Potrich, E.: Expressing workflow patterns for web services: The case of pews. J.UCS **12**(7) (jul 2006) 903–921
19. Ovaska, E., Evesti, A., Henttonen, K., Palviainen, M., Aho, P.: Knowledge based quality-driven architecture design and evaluation. Information & Software Technology **52**(6) (2010) 577–601
20. Papazoglou, M.P., Pohl, K., Parkin, M., Metzger, A., eds.: Service Research Challenges and Solutions for the Future Internet. In Papazoglou, M.P., Pohl, K., Parkin, M., Metzger, A., eds.: S-CUBE Book. Volume 6500 of LNCS., Springer (2010)
21. Salaün, G., Bordeaux, L., Schaerf, M.: Describing and Reasoning on Web Services using Process Algebra. In: Proc. IEEE International Conference on Web Services. ICWS '04, Washington, DC, USA, IEEE Computer Society (2004)
22. Schmeling, B., Charfi, A., Mezini, M.: Composing Non-functional Concerns in Composite Web Services. In: Proc. ICWS 2011. (july 2011) 331 –338
23. Souza Neto, P.A.: A methodology for building service-oriented applications in the presence of non-functional properties. PhD thesis, Federal University of Rio Grande do Norte (2012) Available at http://www3.ifrn.edu.br/∼placidoneto/thesisPlacidoASNeto.pdf.
24. Tongrungrojana, R., Lowe, D.: WIED: A Web Modelling Language for Modelling Architectural-Level Information Flows. J. Digit. Inf. **5**(2) (2004)
25. Van Der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distrib. Parallel Databases **14**(1) (July 2003) 5–51
26. Xiao, H., Chan, B., Zou, Y., Benayon, J.W., O'Farrell, B., Litani, E., Hawkins, J.: A Framework for Verifying SLA Compliance in Composed Services. In: ICWS. (2008)