πSOD-M: A Methodology for Building Service-Oriented Applications in the Presence of Non-Functional Properties

Plácido A. Souza Neto[a], Genoveva Vargas-Solar[b], Martin A. Musicante[c], Valeria de Castro[d], Umberto Costa[c]

[a] *Federal Institute of Rio Grande do Norte – Natal-RN, Brazil*
[b] *Université de Grenoble – Saint Martin d'Hères, France*
[c] *Federal University of Rio Grande do Norte – Natal-RN, Brazil*
[d] *Universidad Rey Juan Carlos – Móstoles, Spain*

Abstract

This paper presents...

Keywords:

1. Introduction

Intro SAC

Functional properties of a computer system are characterized by the effect produced by the system when given a defined input. Functional properties are not the only crucial aspect in the software development process. Other properties need to be addressed to fit in the application with its context. These other aspects are called Non-Functional Properties.

Non-Functional Requirements (NFRs) specify those properties that are not addressed by the functional specification. They are often called *qualities* of the software system. Non-Functional Requirements may specify response time, security constraints or quality of the solution, among others.

Service-Oriented Computing [**?**], is a software development paradigm where pre-existing services are combined to produce more complex applications. The development of service-based applications can benefit from the inclusion of NFRs to the software process from its early stages. Failure to comply with this inclusion means that the final application is obtained from a partial specification, making the deployment a difficult task. 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 their context. Non-functional properties of service oriented applications have been addressed in academic works and standards [**?** **?** 3]. Different proposals [4, 2, 7, 10, 19, 11**?**] support non-functional requirements in the context of web service development.

Most software development methods define software processes that use the notion of refinement. Software process begins with the formulation of an abstract specification, which is successively refined to yield the implementation of the system. Methods for the development of web service applications are no exception to this rule. At least two levels of abstraction can be distinguished: a *Business Level*, including the abstract specification, and a *System Level*, including actual computer programs that implements the system.

In the case of web service applications we will distinguish two separate layers of the implementation. The *Composition Layer* is the upper layer of the implementation. It defines the workflow of the system, in terms of individual service calls. The *Service Layer* defines those services that are called by the composition.

$$\mathbb{O} \diagdown \beth \eth \beth \ltimes \supset \prec$$

Service oriented computing is at the origin of an evolution in the field of software development. An important challenge of service oriented development is to ensure the alignment between IT systems and the business logic. Thus, organizations are seeking for mechanisms to deal with the gap between the systems developed and business needs [5]. The literature stresses the need for methodologies and techniques for service oriented analysis and design, claiming that they are the cornerstone in the development of meaningful services' based applications [14]. In this context, some authors argue that the convergence of model-driven software development, service orientation and better techniques for documenting and improving business processes are the key to make real the idea of rapid, accurate development of software that serves, rather than dictates, software users' goals [18].

Service oriented development methodologies providing models, best practices, and reference architectures to build services' based applications mainly address functional aspects [**?** 6, 8, 15]. Non-functional aspects concerning services' and application's "semantics", often expressed as requirements and constraints in general purpose methodologies, are not fully considered or they are added once the application has been implemented in order to ensure some level of reliability (e.g., data privacy, exception handling, atomicity, data persistence). This leads to services' based applications that are partially specified and that are thereby partially compliant with application requirements.

The objective of this work is to model non-functional constraints and associate them to services' based applications early during the services' composition modeling phase. Therefore this paper presents π-SOD-M, a model-driven method that extends the SOD-M [8] for building reliable services' based information systems (SIS).

Our work proposes to extend the SOD-M [8] method with (i) the notion of *A-Policy* [9] for representing non-functional constraints associated to services' based applications. Our work also (ii) defines the π-PEWS meta-model [17] providing guidelines for expressing the composition and the *A-policies*. Finally, our work (iii) defines model to model transformation rules for generating the π-PEWS model of a reliable services' composition starting from the extended services' composition model; and, model to text transformations for generating the corresponding implementation. As will be shown within our environment implementing these meta models and rules, one may represent both systems' cross-cutting aspects (e.g., exception handling for describing what to do when a service is not available, recovery, persistence aspects) and constraints associated to services, that must be respected for using them (e.g., the fact that a service requires an authentication protocol for executing a method).

The remainder of the paper is organized as follows. Section 2 gives an overview of our approach. It describes a motivation example that integrates and synchronizes well-known social networks services namely Facebook, Twitter and, Spotify. Sections 3, 4, and 5 describe respectively the three key elements of our proposal, namelly the π-SCM and π-PEWS meta-models and the transformation rules that support the semi-automatic generation of reliable services' compositions. Section 6 describes implementation and validation issues. Section 7 analyses related work concerning policy/contract based programming and, services' composition platforms. Section 8 concludes the paper and discusses future work.

2. Modeling reliable services' compositions with π-SOD-M

Given a set of services with their exported methods known in advance or provided by a service directory, building services' based applications can be a simple task that implies expressing an application logic as a services' composition. The challenge being ensuring the compliance between the specification and the resulting application. Software engineering methods (e.g., [**?** 6, 8, 15]) today can help to ensure this compliance, particularly when information systems include several sometimes complex business processes calling Web services or legacy applications exported as services.

2.1. Modeling a services' based application

Figure 1 shows SOD-M that defines a service oriented approach providing a set of guidelines to build services' based information systems (SIS) [8**?**]. Therefore,

SOD-M proposes to use services as first-class objects for the whole process of the SIS development and it follows a Model Driven Architecture (MDA) [13] approach. Extending from the highest level of abstraction of the MDA, SOD-M provides a conceptual structure to: first, capture the system requirements and specification in high-level abstraction models (computation independent models, CIMs); next, starting from such models build platform independent models (PIMs) specifying the system details; next transform such models into platform specific models (PSMs) that bundles the specification of the system with the details of the targeted platform; and finally, serialize such model into the working-code that implements the system. As

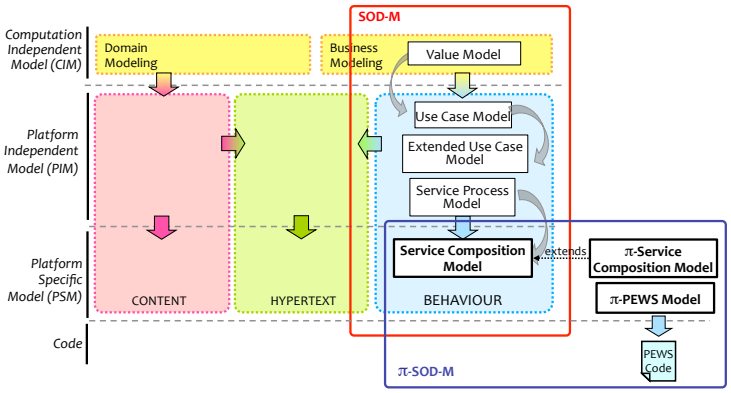

Figure 1: SOD-M development process

shown in Figure 1, the SOD-M model-driven process begins by building the high-level computational independent models and enables specific models for a service platform to be obtained as a result [8]. Referring to the "To Publish Music" application, using SOD-M the designer starts defining an E3value model [1] at the CIM level and then the corresponding models of the PIM are generated leading to a services' composition model (SCM).

2.2. Modeling non-functional constraints of services' based applications

Adding non-functional requirements and services constraints in the services' composition is a complex task that implies programming protocols for instance authentication protocols to call a service in our example, and atomicity (exception handling and recovery) for ensuring a true synchronization of the results produced by the service methods calls.

[1] The E3 value model is a business model that represents a business case and allows to understand the environment in which the services' composition will be placed [**?**].

Service oriented computing promotes ease of information systems' construction thanks, for instance, to services' reuse. Yet, this is not applied to non-functional constraints as the ones described previously, because they do not follow in general the same service oriented principle and because they are often not fully considered in the specification process of existing services' oriented development methods. Rather, they are either supposed to be ensured by the underlying execution platform, or they are programmed through ad-hoc protocols. Besides, they are partially or rarely methodologically derived from the application specification, and they are added once the code has been implemented. In consequence, the resulting application does not fully preserve the compliance and reuse expectations provided by the service oriented computing methods.

Our work extends SOD-M for building applications by modeling the application logic and its associated non-functional constraints and thereby ensuring the generation of reliable services' composition. As a first step in our approach, and for the sake of simplicity we started modeling non-functional constraints at the PSM level. Thus, in this paper we propose the π-SCM, the services' composition meta-model extended with *A-policies* for modeling non-functional constraints (highlighted in Figure 1 and described in Section 3). π-SOD-M defines the π-PEWS meta-model providing guidelines for expressing the services' composition and the *A-policies* (see Section 4), and also defines model to model transformation rules for generating π-PEWS models starting from π-SCM models that will support executable code generation (see Section 5). Finally, our work defines model to text transformation rules for generating the program that implements both the services' composition and the associated *A-policies* and that is executed by an adapted engine (see Section 6).

3. π services' composition meta-model

The *A-policy* based services' composition meta-model (see in Figure 3) represents a workflow needed to implement a services' composition, identifying those entities that collaborate in the business processes (called BUSINESS COLLABORATORS [2]) and the ACTIONS that they perform. This model is represented by means of a UML activity diagram. Thus, as shown in Figure 2, the meta-model includes typical modeling elements of the activity diagram such as ACTIVITYNODES, INITIALNODES and FINALNODES, DECISIONNODES, etc., along with new elements defined by SOD-M such as BUSINESS COLLABORATORS, SERVICEACTIVITY and ACTION (see the white elements in Figure 3).

[2]We use CAPITALS for referring to meta-models' classes.

- A BUSINESS COLLABORATOR element represents those entities that collaborate in the business processes by performing some of the required actions. They are graphically presented as a partition in the activity diagram. A collaborator can be either internal or external to the system being modelled. When the collaborator of the business is external to the system, the attribute IsExternal [3] of the collaborator is set to true.

- ACTION, a kind of EXECUTABLENODE, are represented in the model as an activity. Each action identified in the model describes a fundamental behaviour unit which represents some type of transformation or processing in the system being modelled. There are two types of actions: i) a WebService (attribute Type is WS); and ii) a simple operation that is not supported by a Web Service, called an ACTIVITYOPERATION (attribute Type is AOP).

- The SERVICEACTIVITY element is a composed activity that must be carried out as part of a business service and is composed of one or more executable nodes.

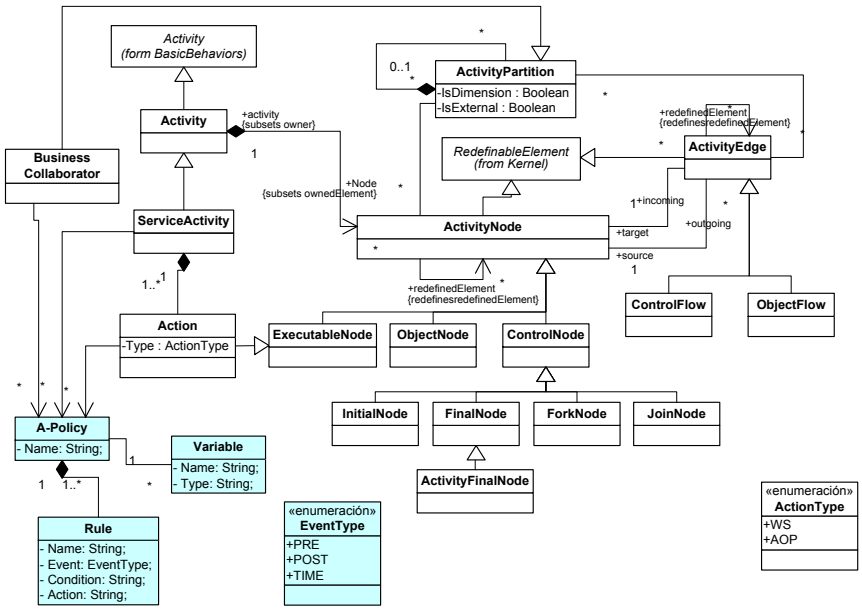

Figure 2: *A-policy* based services' composition meta-model (π-SCM)

[3]We use the sans serif font for referring to models' classes defined using a meta-model.

To illustrate the use of the π-SCM meta-model we used it for defining the *A-policy* based composition model of the "To Publish Music" scenario (see Figure 3). There are three external business collaborators (*Spotify, Twitter* and *Facebook* [4]). It also shows the business process of the "To Publish Music" application that consists of three service activities: *Listen Music*, *Public Music* and *Confirmation*. Note that the action *Publish Music* of the application calls the actions of two service collaborators namely *Facebook* and *Twitter*.

Instead of programming different protocols within the application logic, we propose to include the modeling of non-functional constraints like transactional behaviour, security and adaptability at the early stages of the services' composition engineering process. We model non-functional constraints of services' compositions using the notion of *A-policy* [9**?**], a kind of pattern for specifying *A-policy* types. In order to represent constraints associated to services compositions, we extended the SOD-M services' composition model with two concepts: RULE and A-POLICY (see blue elements in the π-SCM meta-model in Figure 2).

The RULE element represents an event - condition - action rule where the EVENT part represents the moment in which a constraint can be evaluated according to a condition represented by the CONDITION part and the action to be executed for reinforcing it represented by the ACTION part. An *A-policy* groups a set of rules. It describes global variables and operations that can be shared by the rules and that can be used for expressing their Event and Condition parts. An *A-Policy* is associated to the elements BUSINESSCOLLABORATOR, SERVICEACTIVITY and, ACTION of the π-SCM meta-model (see Figure 2) .

Given that *Facebook* and *Twitter* services require authentication protocols in order to execute methods that will read and update the users' space. A call to such services must be part of the authentication protocol required by these services. In the example we associate two authentication policies, one for the open authentication protocol, represented by the class Twitter OAuthPolicy that will be associated to the activity UpdateTwitter (see Figure 3). In the same way, the class Facebook HTTPAuthPolicy, for the http authentication protocol will be associated to the activity UpdateFacebook. OAuth implements the open authentication protocol. As shown in Figure 3, the *A-policy* has a variable Token that will be used to store the authentication token provided by the service. This variable type is imported through the library OAuth.Token. The *A-policy* defines two rules, both can be triggered by events of type ActivityPrepared: (i) if no token has been associated to the variable token, stated in by the condition of rule

[4]We use *italics* to refer to concrete values of the classes of a model that are derived from the classes of a meta-model.

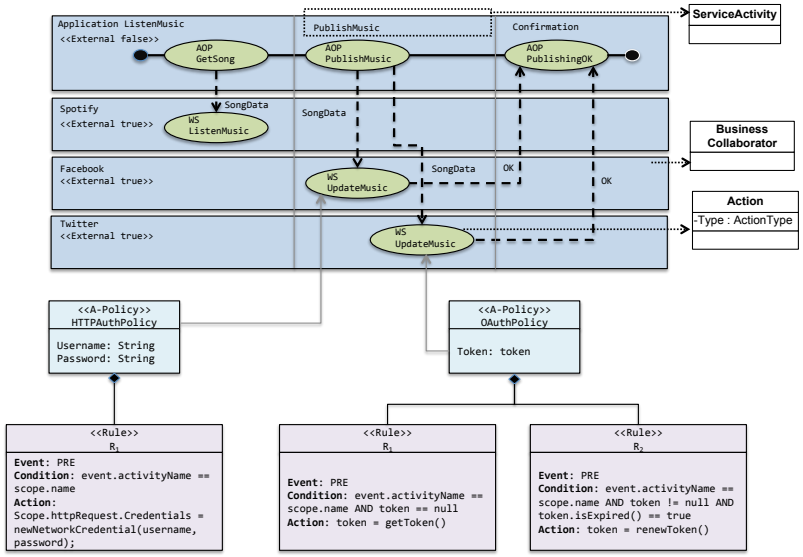

Figure 3: Services' composition model for the business service "To publish music"

R_1, then a token is obtained (action part of R_1); (ii) if the token has expired, stated in the condition of rule R_2, then it is renewed (action part of R_2). Note that the code in the actions profits from the imported OAuth.Token for transparently obtaining or renewing a token from a third party.

HTTP-Auth implements the HTTP-Auth protocol. As shown in Figure 3, the *A-policy* imports an http protocol library and it has two variables username and password. The event of type ActivityPrepared is the triggering event of the rule R_1. On the notification of an event of that type, a credential is obtained using the username and password values. The object storing the credential is associated to the scope, i.e., the activity that will then use it for executing the method call.

Thanks to rules and policies it is possible to model and associate non-functional properties to services' compositions and then generate the code. For example, the atomic integration of information retrieved from different social network services, automatic generation of an integrated view of the operations executed in different social networks or for providing security in the communication channel when the payment service is called.

Back to the definition process of a SIS, once the *A-policy* based services' composition model has been defined, then it can be transformed into a model (i.e., π-PEWS model) that can support then executable code generation. The following Section describes the π-PEWS meta-model that supports this representation.

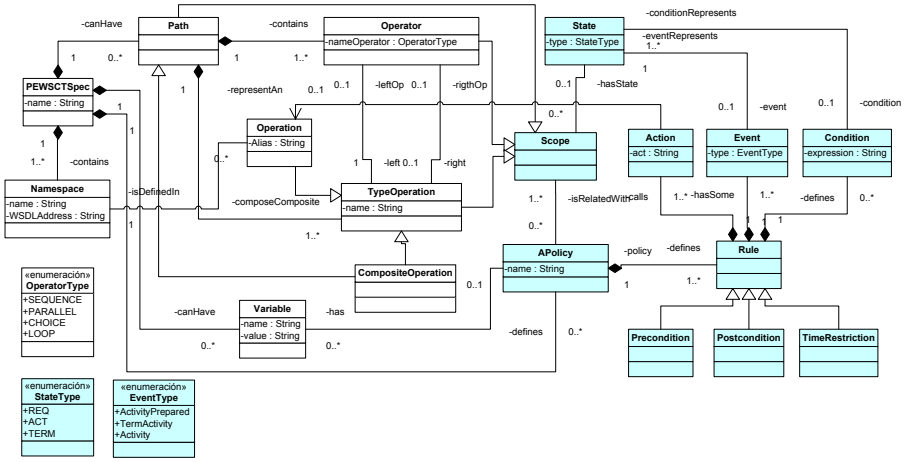

Figure 4: π-PEWS Metamodel

4. π-PEWS meta-model

The idea of the π-PEWS meta-model is based on the services' composition approach provided by the language PEWS[**?** 17] (*Path Expressions for Web Services*), a programming language that lets the service designer combine the methods or subprograms that implement each operation of a service, in order to achieve the desired application logic. Figure 4 presents the π-PEWS meta-model consisting of classes representing:

- A services' composition: NAMESPACE representing the interface exported by a service, OPERATION that represents a call to a service method, COMPOSITEOPERATION, and OPERATOR for representing a services' composition and PATH representing a services' composition. A PATH can be an OPERATION or a COMPOUND OPERATION denoted by an identifier. A COMPOUND OPERATION is defined using an OPERATOR that can be represent sequential (.) and parallel ($\|$) composition of services, choice (+) among services, the sequential ($*$) and parallel ($\{\ldots\}$) repetition of an operation or the conditional execution of an operation ($[C]S$).

- *A-Policies* that can be associated to a services' composition: A-POLICY, RULE, EVENT, CONDITION, ACTION, STATE, and SCOPE.

As shown in the diagram an A-POLICY is applied to a SCOPE that can be either an OPERATION (e.g., an authentication protocol associated to a method exported by a service), an OPERATOR (e.g., a temporal constraint associated to a sequence of

operators, the authorized delay between reading a song title in Spotify and updating the walls must be less then 30 seconds), and a PATH (e.g., executing the walls' update under a strict atomicity protocol – all or noting). It groups a set of ECA rules, each rule having a classic semantics, i.e, *when an event of type E occurs if condition C is verified then execute the action A*. Thus, an *A-policy* represents a set of reactions to be possibly executed if one or several triggering events of its rules are notified.

- The class SCOPE represents any element of a services' composition (i.e., operation, operator, path).
- The class A-POLICY represents a recovery strategy implemented by ECA rules of the form EVENT - CONDITION - ACTION. A *A-policy* has variables that represent the view of the execution state of its associated scope, that is required for executing the rules. The value of a variable is represented using the type VARIABLE. The class A-POLICY is specialized for defining specific constraints, for instance authentication *A-policies*.

Given a π-SCM model of a specific services' based application (expressed according to the π-SCM meta-model), it is possible to generate its corresponding π-PEWS model thanks to transformation rules. The following Section describes the transformation rules between the π-SCM and π-PEWS meta-models of our method.

5. Transformation rules

Figure 5 shows the transformation principle between the elements of the π-SCM meta-model used for representing the services' composition into the elements of the π-PEWS meta-model. There are two groups of rules: those that transform services' composition elements of the π-SCM to π-PEWS meta-models elements; and those that transform rules grouped by policies into *A-policy* types.

5.1. Transformation of the services' composition elements of the π-SCM to the π-PEWS elements

A named action of the π-SCM represented by *Action* and *Action:name* is transformed to a named class OPERATION with a corresponding attribute name OPERATION:NAME. A named service activity represented by the elements *ServiceActivity* and *ServiceActivity:name* of the π-SCM, are transformed into a named operation of the π-PEWS represented by the elements COMPOSITEOPERATION and COMPOSITEOPERATION:NAME. When more than one action is called, according to the following composition patterns expressed using the operators *merge, decision, fork and join* in the π-SCM the corresponding transformations, according to the PEWS operators presented above, are (see details in Figure 5):

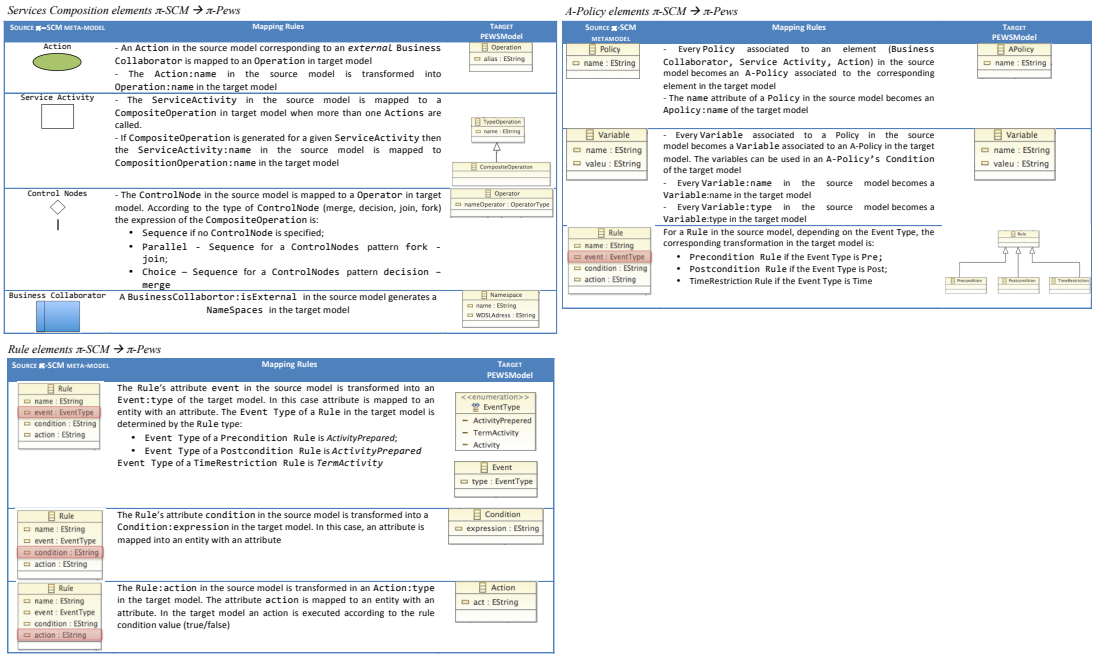

Figure 5: π-SCM to π-PEWS transformation

- $op_1.op_2$ if no *ControlNode* is specified
- $(op_1 \parallel op_2).op_3$ if control nodes of type *fork, join* are combined
- $(op_1 + op_2).op_3$ if control nodes of type *decision, merge* are combined

In the scenario "To Publish Music" the service activity PublishMusic of the π-SC model specifies calls to two Activities of type *UpdateMusic*, respectively concerning the Business Services *Facebook* and *Twitter*. Given that no ConstrolNode is specified by the π-SC model, the corresponding transformation is the expression that defines a Composite Operation named *PublishSong* of the π-PEWS model of the form (PublishFacebook $\parallel$ PublishTwitter).

5.2. Transformation of rules grouped by A-policies in the π-SCM to A-Policies of π-PEWS

The *A-policies* defined for the elements of the π-SCM are transformed into A-POLICY classes, named according to the names expressed in the source model. The transformation of the rules expressed in the π-SCM is guided by the event types associated to these rules. The variables associated to an *A-policy* expressed in the π-SCM

as <*Variable:name, Variable:type*> are transformed into elements of type VARIABLE with attributes NAME and TYPE directly specified from the elements *Variable:name* and *Variable:type* of the π-SCM model.

As shown in Figure 5, for an event of type *Pre* the corresponding transformed rule is of type PRECONDITION; for an event of type *Post* the corresponding transformed rule is of type POSTCONDITION; finally, for an event of type *TimeRestriction* the corresponding transformed rule is of type TIME. The condition expression of a rule in the π-SCM (*Rule:condition*) is transformed into a class *Condition:expression* where the attributes of the expression are transformed into elements of type ATTRIBUTE.

In the scenario "To Publish Music" the Policies *OAuthPolicy* and *HTTPAuthPolicy* of the π-SCM model are transformed into *A-policies* of type Precondition of the π-PEWS model of the scenario. Thus in both cases the events are of type ActivityPrepared. These policies, as stated in the π-SCM model, are associated to Activities. In the corresponding transformation they are associated to Operations *PublishFacebook* and *PublishTwitter*.

6. Implementation issues

This section describes the π-SOD-M development environment that implements the generation of *A-policies*' based services' compositions. For a given services' based application, the process consists in generating the code starting from a π-SCM modeling an application. Note that the services' composition model is not modeled from scratch, but it is the result of a general process defined by the π-SOD-M method in which a set of models are built following a service oriented approach [8].

6.1. π-SOD-M Development Environment

Figure 6 depicts a general architecture of the π-SOD-M Development Environment showing the set of plug-ins developed in order to implement it. The environment implements the abstract architecture shown in Figure 1. Thus, it consists of plug-ins implementing the π-SCM and π-PEWS meta-models used for defining models specifying services' compositions and their associated policies; and ATL rules for transforming PSM models (model to model transformation) and finally generating code (model to text transformation).

- We used the Eclipse Modeling Framework (EMF) [5] for implementing the meta-models π- SCM and π-PEWS. Then, starting form these meta-models, we

[5]The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model.

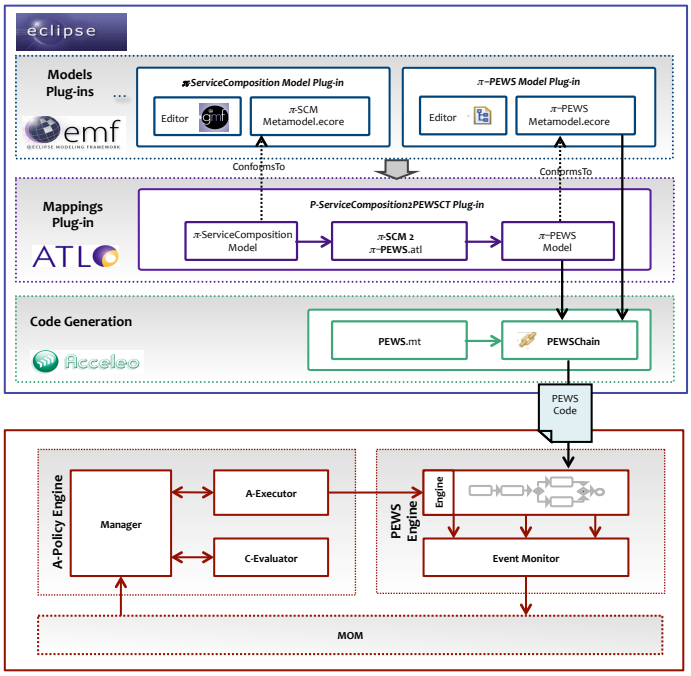

Figure 6: π-SOD-M Development Environment

developed the models' plug-ins needed to support the graphical representation of the π- SCM and π-PEWS models (π-ServiceCompostion Model and π-PEWS Model plug-ins).

- We used ATL [6] for developing the mapping plug-in implementing the mappings between models (π-ServiceComposition2π-PEWS Plug-in).

- We used Acceleo [7] for implementing the code generation plug-in. We coded the pews.mt program that implements the model to text transformation for generating executable code. It takes as input a π-PEWS model implementing a specific services' composition and it generates the code to be executed by the *A-policy* based services' composition execution environment.

As shown in Figure 6, once an instance of a PEWS code is obtained starting form a particular π-services' composition model it can be executed over *A-policy* based services' composition execution environment consisting of a composition engine and a *A-policy* manager. The *A-policy* manager consists of three main components Manager, for scheduling the execution of rules, C-Evaluator and A-Executor respectively for evaluating rules' conditions and executing their actions. The *A-policy* Manager interacts with a composition engine thanks to a message communication layer (MOM).

The composition engine manages the life cycle of the composition. Once a composition instance is activated, the engine schedules the composition activities according to the composition control flow. Each activity is seen as the process where the service method call is executed. The execution of an activity has four states: prepared, started, terminated, and failure. The execution of the control flow (sequence, and/or split and join) can also be prepared, started, terminated and raise a failure.

At execution time, the evaluation of policies done by the *A-policy* manager must be synchronized with the execution of the services' composition (i.e., the execution of an activity or a control flow). Policies associated to a scope are activated when the execution of its scope starts. A *A-policy* will have to be executed only if one or several of its rules is triggered. If several rules are triggered the *A-policy* manager first builds an execution plan that specifies the order in which such rules will be executed according to the strategies defined in the following section. If rules belonging to several policies are triggered then policies are also ordered according to an execution

[6] http://eclipse.org/atl/. An ATL program is basically a set of rules that define how source model elements are matched and navigated to create and initialize the elements of the target models.

[7] http://www.acceleo.org/pages/home/en

plan. The execution of policies is out of the scope of this paper, the interested reader can refer to [9] for further details.

6.2. Validation

We validated our approach by implementing the "To Publish Music" application. It consists in atomically synchronizing the status of a user's accounts according to the music she listens in Spotify and using authentication protocols for updating the walls' status in her Facebook and Twitter accounts. Once the services' composition model has been annotated with the corresponding policies, then the reliable composition code can be generated using the rules defined for this purpose. Our implementation includes a code generator that generates executable code for a reliable services' composition. The model represents an services' composition expression and its associated policies.

We tested the application in order to see whether the coordination tolerated services unavailability keeping the status synchronized. Particularly, the token of the activity associated to Twitter was consistently obtained when the service was available. We also implemented a reference coordination without policies. We dealt with the availability exceptions by hand and the authentication protocols were embedded within the activities code. During our validation Twitter changed the authentication protocol and the maintenance of the application this implied re-programming the reference application. Instead, with the policies approach we only had to desactivate the corresponding policy and associate the appropriate one to the activity calling the service Twitter (one code line), and run the application again.

7. Related works

Related works to our approach include standards devoted for expressing non-functional constraints for services and services' compositions. They also include methods and approaches for modeling non-functional constraints.

7.1. Programming non-functional properties for services

Current standards in services' composition implement functional, non-functional constraints and communication aspects by combining different languages and protocols. WSDL and SOAP among others are languages used respectively for describing services' interfaces and message exchange protocols for calling methods exported by such services. For adding a transactional behaviour to a services' composition it is necessary to implement WS-Coordination, WS-Transaction, WS-BussinessActivity and WS-AtomicTransaction. The selection of the adequate protocols for adding a

specific non-functional constraints to a services' composition (e.g., security, transactional behaviour and adaptability) is responsibility of a programmer. As a consequence, the development of an application based on a services' composition is a complex and a time-consuming process. This is opposed to the philosophy of services that aims at facilitating the integration of distributed applications. Other works, like [**?**] introduce a model for transactional services composition based on an advanced transactional model. [**?**] proposes an approach that consists of a set of algorithms and rules to assist designers to compose transactional services. In [**?**] the model introduced in [**?**] is extended to web services for addressing atomicity.

7.2. Modeling non-functional properties

There are few methodologies and approaches that address the explicit modeling of non functional properties for service based applications. Software process methodologies for building services based applications have been proposed in[15**?** **?** , 16], and they focus mainly on the modeling and construction process of services based business processes that represent the application logic of information systems.

Design by Contract [**?**] is an approach for specifying web services and verifying them through runtime checkers before they are deployed. A contract adds behavioral information to a service specification, that is, it specifies the conditions in which methods exported by a service can be called. Contracts are expressed using the language *jmlrac* [12]. The *Contract Definition Language* (CDL) [**?**] is a XML-based description language, for defining contracts for services. There are an associated architecture framework, design standards and a methodology, for developing applications using services. A services' based application specification is generated after several [1] B-machines refinements that describe the services and their compositions. [15] proposes a methodology based on a SOA extension. This work defines a service oriented business process development methodology with phases for business process development. The whole life-cycle is based on six phases: planning, analysis and design, construction and testing, provisioning, deployment, and execution and monitoring.

7.3. Discussion

As WS-* and similar approaches, our work enables the specification and programing of crosscutting aspects (i.e., atomicity, security, exception handling, persistence). In contrast to these approaches, our work specifies policies for a services' composition in an orthogonal way. Besides, these approaches suppose that non-functional requirements are implemented according a the knowledge that a programmer has of a specific application requirements but they are not derived in a methodological

way, leading to ad-hoc solutions that can be difficult to reuse. In our approach, once defined *A-Policies* for a given application they can be reused and/or specialized for another one with the same requirements or that uses services that impose the same constraints.

Furthermore, unlike methodologies and approaches providing best practices presented above, the main contribution of our proposal is that, integrated to a method that proposes meta-models at different levels (CIM, PIM and PSM) and extending the PSM meta-models, it enables the design and development of services' based applications that can be reused and that are reliable.

8. Conclusions and future work

This paper presented π-SOD-M for specifying and designing reliable service based applications. We model and associate policies to services' based applications that represent both systems' cross-cutting aspects and use constraints stemming from the services used for implementing them. We extended the SOD-M method, particularly the π-SCM (services' composition meta-model) and π-PEWS meta-models for representing both the application logic and its associated non-functional constraints and then generating its executable code. We implemented the meta-models on the Eclipse platform and we validated the approach using a use case that uses authentication policies.

Non-functional constraints are related to business rules associated to the general "semantics" of the application and in the case of services' based applications, they also concern the use constraints imposed by the services. We are currently working on the definition of a method for explicitly expressing such properties in the early stages of the specification of services based applications. Having such business rules expressed and then translated and associated to the services' composition can help to ensure that the resulting application is compliant to the user requirements and also to the characteristics of the services it uses.

Programming non-functional properties is not an easy task, so we are defining a set of predefined *A-policy* types with the associated use rules for guiding the programmer when she associates them to a concrete application. *A-policy* type that can also serve as patterns for programming or specializing the way non-functional constraints are programmed.

References

[1] J. R. Abrial, M. K. O. Lee, D. Neilson, P. N. Scharbach, and I. H. Sorensen. The b-method. In Soren Prehn and W. J. Toetenel, editors, *VDM Europe (2)*,

volume 552 of *Lecture Notes in Computer Science*, pages 398–405. Springer, 1991.

[2] S. Agarwal, S. Lamparter, and R. Studer. Making web services tradable: A policy-based approach for specifying preferences on web service properties. *J. Web Sem.*, 7(1):11–20, 2009.

[3] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek. Web service choreography interface. Technical report, World Wide Web Consortium, 2002.

[4] S. M. Babamir, S. Karimi, and M. R. Shishechi. A broker-based architecture for quality-driven web services composition. In *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, pages 1 –4, dec. 2010.

[5] M. Bell. *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. 2008.

[6] A. Brown. SOA Development Using the IBM Rational Software Development Platform: A Practical Guide. In *Rational Software*, 2005.

[7] S. Chollet and P. Lalanda. An extensible abstract service orchestration framework. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 831–838, Washington, DC, USA, 2009. IEEE Computer Society.

[8] V. de Castro, E. Marcos, and R. Wieringa. 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. *International Journal of Cooperative Information Systems*, 18(2), 2009.

[9] J. A. Espinosa-Oviedo, G. Vargas-Solar, J. L. Zechinelli-Martini, and C. Collet. Policy driven services coordination for building social networks based applications. In *In Proc. of the 8th Int. Conference on Services Computing (SCC'11), Work-in-Progress Track*, Washington, DC, USA, July 2011. IEEE.

[10] C. Gutiérrez, D. G. Rosado, and E. Fernández-Medina. The practical application of a process for eliciting and designing security in web service systems. volume 51, pages 1712–1738, Newton, MA, USA, December 2009. Butterworth-Heinemann.

[11] B. Jeong, H. Cho, and C. Lee. On the functional quality of service (fqos) to discover and compose interoperable web services. *Expert Syst. Appl.*, 36(3):5411–5418, April 2009.

[12] G. T. Leavens, Y. Cheon, C. Clifton, C. Ruby, and D. R. Cok. How the design of jml accomodates both runtime assertion checking and formal verification. In *FMCO*, pages 262–284, 2002.

[13] J. Miller and J. Mukerji. Mda guide. 2003.

[14] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11), 2007.

[15] M. P. Papazoglou and W. J. Heuvel, Van Der Heuvel. Service-oriented design and development methodology. *Int. J. Web Eng. Technol.*, 2(4):412–442, July 2006.

[16] E. Ramollari, D. Dranidis, and A. J. H. Simons. A survey of service oriented development methodologies.

[17] P. A. Souza Neto, M. A. Musicante, G. Vargas-Solar, and J. L. Zechinelli-Martini. Adding Contracts to a Web Service Composition Language. *LTPD - 4th Workshop on Languages and Tools for Multithreaded, Parallel and Distributed Programming*, September 2010.

[18] A. Watson. A brief history of MDA, 2008.

[19] H. Xiao, B. Chan, Y Zou, J. W. Benayon, B. O'Farrell, E. Litani, and J. Hawkins. A framework for verifying sla compliance in composed services. In *ICWS*, pages 457–464, 2008.