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 Technological 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

Model Driven Development (MDD) is a top-down approach for the development of software systems. The main ideas of MDD were originally proposed by the Object Management Group (OMG) [**?**], as a set of guidelines for the structuring of specifications. The technique advocates for the use of *models* to specify a software system at different levels of abstraction (called *viewpoints*):

Computation Independent Models (CIM): This level focusses on the environment of the system, as well as on its business and requirement specifications. This viewpoint represents the software system at its highest level of abstraction. At this moment of the development, the structure and system processing details are still unknown or undetermined.

Platform Independent Models (PIM): This level focusses on the system functionality, hiding the details of any particular platform. The specification defines those parts of the system that do not change from one platform to another.

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

Besides the notion of model at each level of abstraction, MDD requires the use of *model transformations* within and between levels. Intra-level transformations are used to provide a unified representation of concepts of a given level. Inter-level transformation implement a refinement process between levels. Transformations may be automatic or semi-automatic.

MDD techniques has been successfully used for the development of hardware and software systems [**?**]. In particular, we are interested in the application of MDD to the design and implementation of web service applications.

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 the requirements imposed by the business logic and the IT systems actually developed. (Moreover, IT systems need to evolve according to the business needs.) Thus, organizations are seeking for mechanisms to bridge the gap between the systems developed and business needs [**?**]. 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 service based applications [**?**]. In this context, we argue that the convergence of model-driven software development, service orientation and better techniques for documenting and improving business processes are key to make real the idea of rapid, accurate development of software that serves, rather than dictates the needs of its users [**?**].

In Service-Oriented Computing, 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 [**?** **?** **?** **?**]. (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 aspects of the services, often expressed as requirements and constraints in general purpose methodologies, are not usually considered from the beginning of the (service) software process. Most methods consider them only after 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 service based applications that are partly specified and, thereby, partly compliant with the requirements of the application. Ideally, non-functional requirements should 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.

In this work, we are interested in the extension of the *Service Oriented Development Method* (SOD-M) [**?**], to support non-functional aspects, from the early stages of software development. SOD-M is aligned with the MDD directives and proposes models, practices and techniques for the development of service-based applications. SOD-M does not provide support for the specification of non-functional requirements, such as security, reliability, and efficiency.

The main goals of our work are:

(i) To propose a methodology for supporting the construction of service-oriented applications, taking into account both functional and non-functional requirements;

(ii) To improve the construction process by providing an abstract view of the application and ensure the conformance to its specification;

(iii) To reduce the programming effort through the semi-automatic generation of models for the application, to produce concrete implementations from high abstraction models;

The rest of the paper is organized as follows. Section...

THE NEXT TWO SECTIONS NEED TO BE RELOCATED –MARTIN

2. SOD-M

The Service-Oriented Development Method (SOD-M) [**?**] is a MDD approach for service-based applications. SOD-M provides models and notation to specify service applications at different levels of abstraction. As usual for MDD, SOD-M metamodels 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*. EXPLAIN THESE TWO HERE –MARTIN.

Models at the PIM level describe the structure of the application flow, while, the PSM level provides transformations towards more specific platforms. The PIM-level models proposed by SOD-M are:

use case:

extended use case:

service process:

service composition.

SOD-M defines three meta-models at the PSM level: *web service interface*, *extended composition service* and *business logic*. These three levels have no support for describing non-functional requirements.

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.

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 application requirements. In this way, SOD-M aims to simplify the design of service-oriented applications, as well as its implementation using current technologies.

3. πSOD-M

πSOD-M provides an environment for building service compositions considering their non-functional requirements. πSOD-M extends the SOD-M meta-models by adding the concept of *Policy* [**?**] to represent 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 models represent both the functional aspects of the application as well as its non-functional constraints. Constraints are restrictions that must be verified during the execution of the application. An example of this is the requirement of the user's authentication for executing some system functions.

Similarly to SOD-M, our approach targets the construction of service-oriented applications that implement business processes. πSOD-M proposes a development process based on the definition of models (instances of the meta-modes) and transformations between models. There are two kinds of transformations: Model-to-model transformations are used during the software process to refine the specification. Model-to-text transformations are the last step of the process and generate code.

We extended SOD-M to include non-functional specifications. Our method defines four meta-models: *π-UseCase*, *π-ServiceProcess*, *π-ServiceComposition* and *π-PEWS*. The former three are extensions of SOD-M meta-models and belong to the PIM level. The *π-PEWS* meta-model is a PSM (Figure 1).

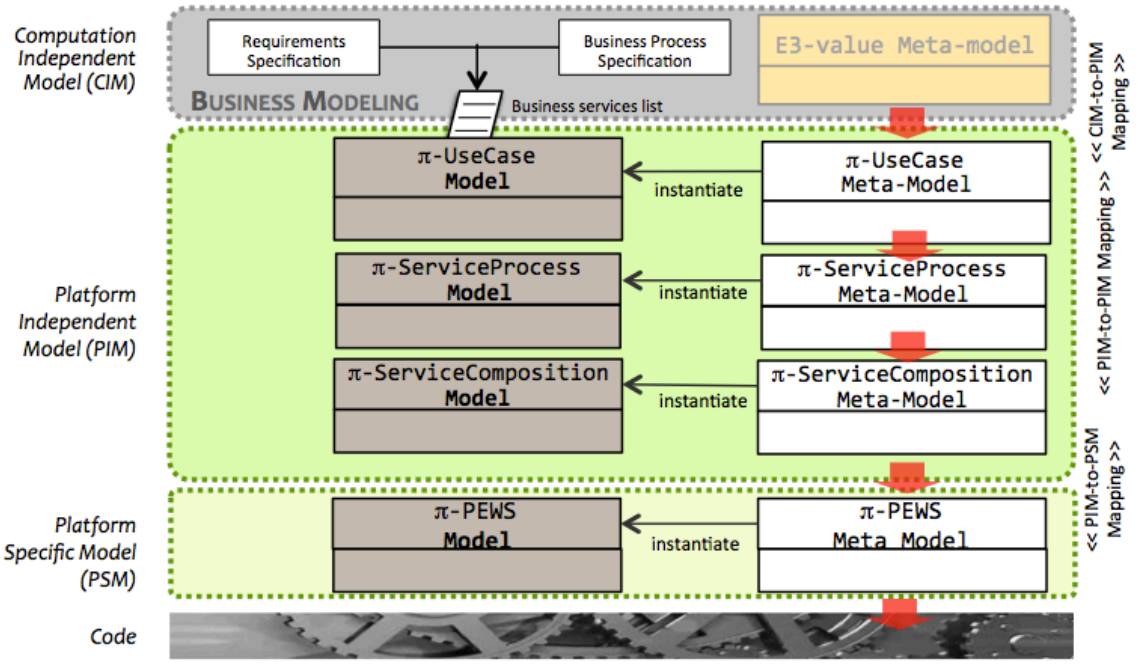

Figure 1: πSOD-M.

The π-*UseCase* meta-model describes functional and non-functional requirements. Non-functional requirements are defined as *constraints* over processing 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 of *Policy* 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.

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. π-*PEWS* models are textual descriptions of service compositions that can be translated into PEWS code [**?** **?**]. Although PEWS is our language of choice, other composition languages can be used as target. This can be accomplished by defining: *(i)* a model-to-model transformation, from a π-*ServiceComposition* model to the corresponding PSM, and *(ii)* a model-to-text transformation, from the this PSM to the composition language.

In the next section we develop an example, to serve as a proof-of-concept. The example will show the actual notation used for models.

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

In order to introduce the context of our work consider the Tracking crime application where civil population and by police share information about criminality in given zones of an imaginary city. Users signal crimes using twitter and police officers notify crimes they have to deal with. Some of this information, if it is not confidential, can be shared to the community of users using this application. Users can track crimes in given zones and see information located in a map according to their privileges. For example civilians can ask Locate the crimes done the last month 100 km from my current position that happened between 8:00 and 14:00. While police officers can ask Locate the regions in my sector where murders happened in the last month This information can come from the police database and from twitter posts. The zones of the city are thereby according to their degree of criminality.

In order to provide these functions the application benefits from existing services that provide information, storage and data visualization functions. Thus, the application is a service based application that implements a business process. The business process is specified in terms of ordered tasks and define the application logic of a Service oriented Application. Tasks can be performed by a person or an entity. In our context, tasks are implemented by Services. Service is an application implemented by a provider that exports an API through a network (e.g., Internet) API is defined using an interface definition language (IDL, e.g., WSDL for Web services). In our example the business procces can start with one of two tasks: Notify a crime, or track a crime. A notified crime can be then stored in a database. Tracked crimes are visualized in a map and then the used can ask for detailed information. They are implemented by four services: twitter and an adhoc police service for notifying crimes, Amazon as persistence service and google maps for visualizing and locating crimes in a map. Note that a task can be implemented by a service composition. This is the case of the task Notify crime in our example that enables to notify crimes through twitter or through the police notification service.

Business processes have also associated rules and constraints that define their non functional requirements. NFR represents the semantics and the conditions in which the tasks must be done. In our example we have some constraints.

- Twtter requires to be accessed through an authentication protocol, and the police crime notification service has a control access protocol and allows only 3 tries to access it.

- Only users with enough privileges can store crimes notifications.
- Users can only track crimes notified by providers t that hey have authorization to contact; For example civil population cannot track all the crimes notified by the police.
- If the google map is unavailable the results of a track request are delivered on text.
- If a user tracks crimes that are provided by a service for which she does not have authorization then show an empty map.
- Detailed information about a crime depends on the user privileges, on the police assignment regions.
- The tracking crime app keeps track of the accesses to detailed information about crimes.

Through the example we underlined that every application implements functional aspects that describe its application logic. Recall that an application logic refers to routines that perform the activities to reach the application objective. Also there are non functional properties derived from NFR. They refer to strategies to be considered for the application execution like: security, isolation, adaptability, atomicity, and more. These non functional properties must be ensured at execution time, and they are not completely defined within the application logic.

The challenge is to define them and to associate them with the application logic considering that different to existing solutions that suppose that it is possible to access the execution stat of all the components of an application and that the application has complete control on them, in the case for service oriented applications the components are autonomous services API does not necessarily export information about methods dependency (e.g., in the REST protocol); they do not share their state (stateless).

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., [**? ? ? ?**]) 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.

4.1. Modeling a services' based application

Figure 2 shows SOD-M that defines a service oriented approach providing a set of guidelines to build services' based information systems (SIS) [**?** **?**]. 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) [**?**] 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

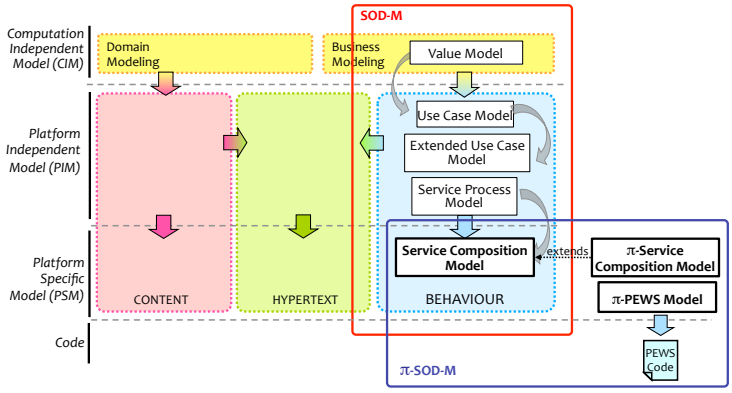

Figure 2: SOD-M development process

shown in Figure 2, 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 [**?**]. 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).

4.2. 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 authenti-

[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 [**?**].

cation 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.

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. In order to do, our work organizes non-functional constraints into three layers representing: application modeling, services composition and services. The service composition layer serves as an integration layer between the services layer that exports methods and has associated constraints and characteristics; and the application layer that expresses requirements. At the application layer NFP can refer to business rules and values. A value NFR expresses constraints about the way data and functions can be accessed and executed. For example accessing methods under security protocols.

Business NFP at the service layer concerns properties that are associated to services and defines how to call their exported operations (business properties). For example, response time, storage capacity (e.g., Dropbox service provides 5Giga free storage). Value constraints concern more on the conditions in which services can be used. For example, accessing to a function within an authentica-tion protocol.

Finally, at the service composition layer gives an abstract view of the kind of properties exported by services that can be combined for providing NFP for a composition. For example, confidentiality, authentication, privacy and access control can provide security at the service composition layer.

As a first step in our approach, 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 2 and described in Section 5). π-SOD-M defines the π-PEWS meta-model providing guidelines for expressing the services' composition and the

A-policies (see Section 6), 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 7). 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 8).

5. π services' composition meta-model

The *A-policy* based services' composition meta-model (see in Figure 4) 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 3, 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 4).

- 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.

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

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

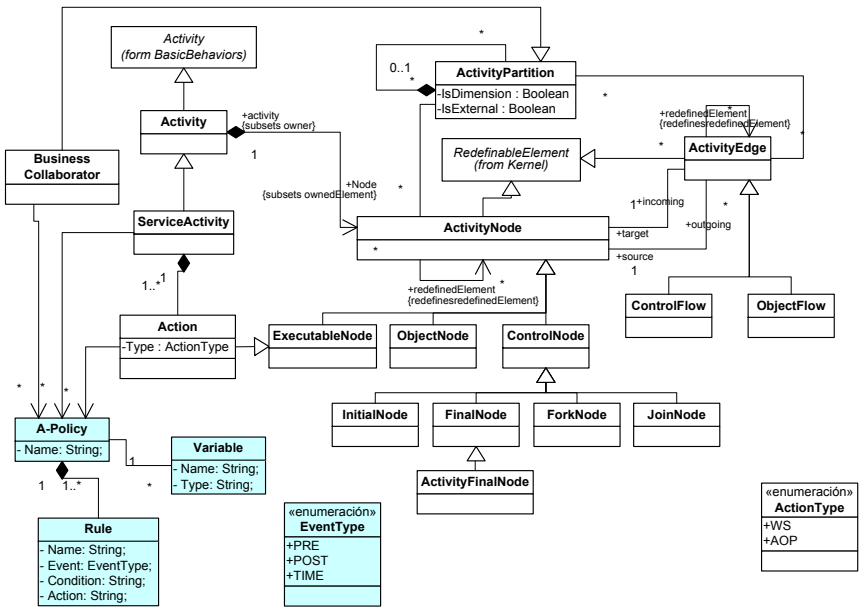

Figure 3: *A-policy* based services' composition meta-model (π-SCM)

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 4). 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* [**?** **?**], 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 3).

[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.

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 3) .

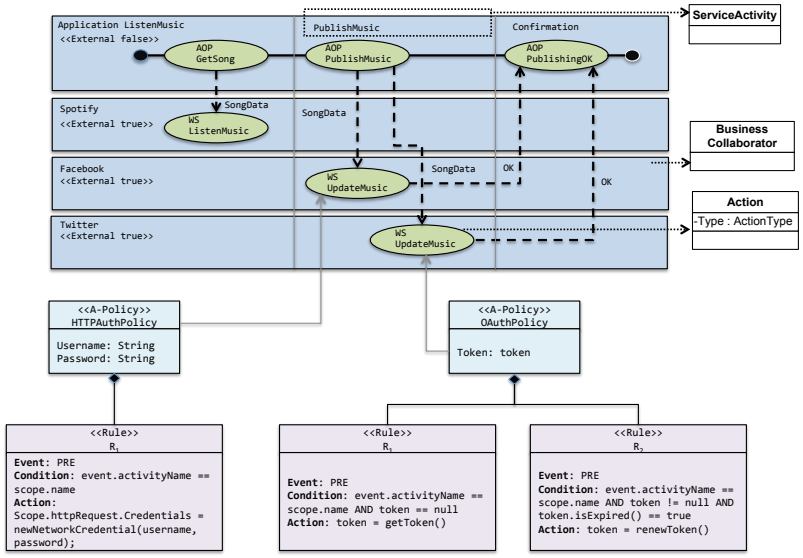

Figure 4: Services' composition model for the business service "To publish music"

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 4). 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 4, 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 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 4, 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.

6. π-PEWS meta-model

The idea of the π-PEWS meta-model is based on the services' composition approach provided by the language PEWS[**?** **?**] (*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 5 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 ($[C]S$).

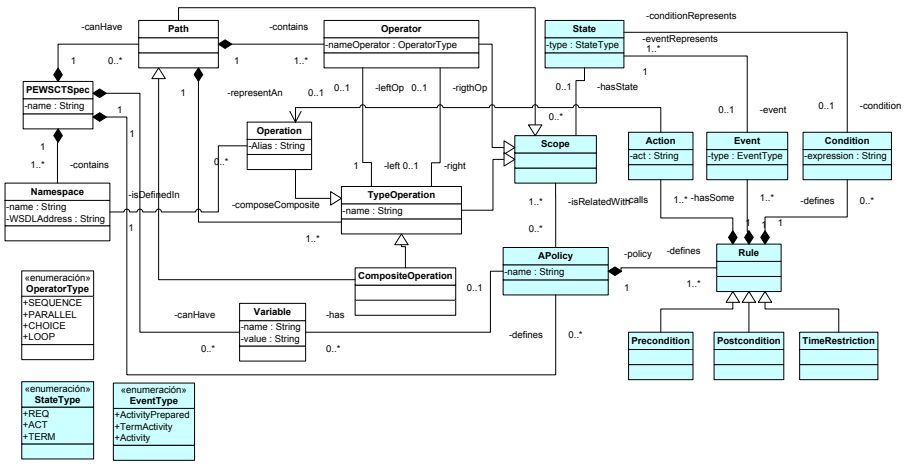

Figure 5: π-Pews Metamodel

- *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.

7. Transformation rules

Figure 6 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.

7.1. 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 6):

- $op_1.op_2$ 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).

7.2. 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

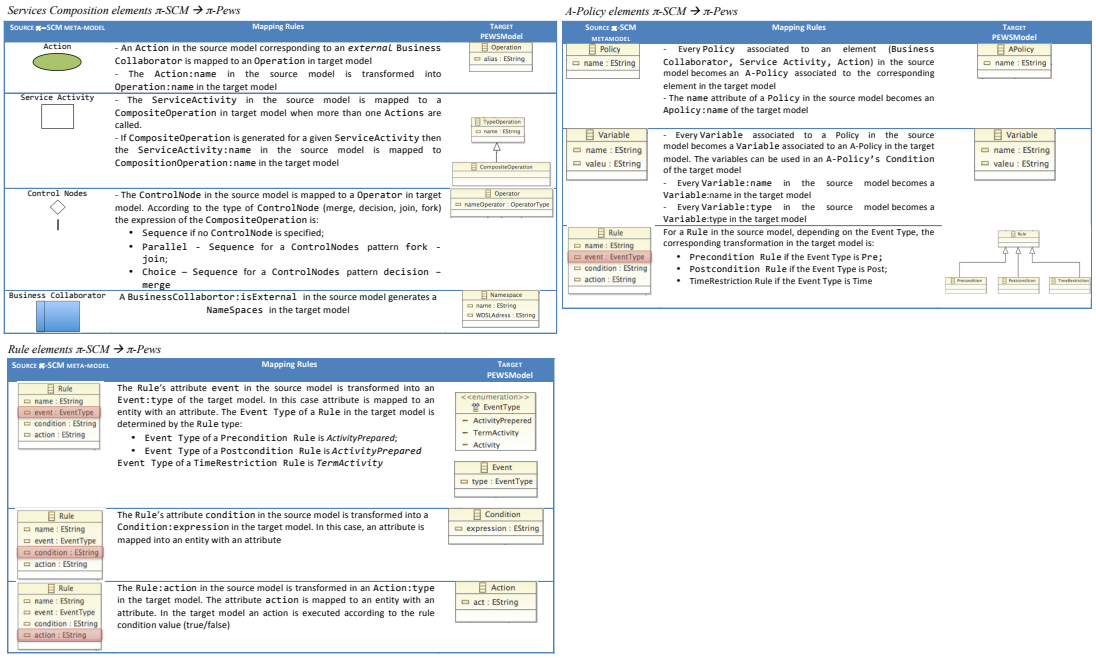

Figure 6: π-SCM to π-PEWS transformation

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 6, 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*.

8. 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 [**?**].

8.1. π-SOD-M Development Environment

Figure 7 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 2. 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).

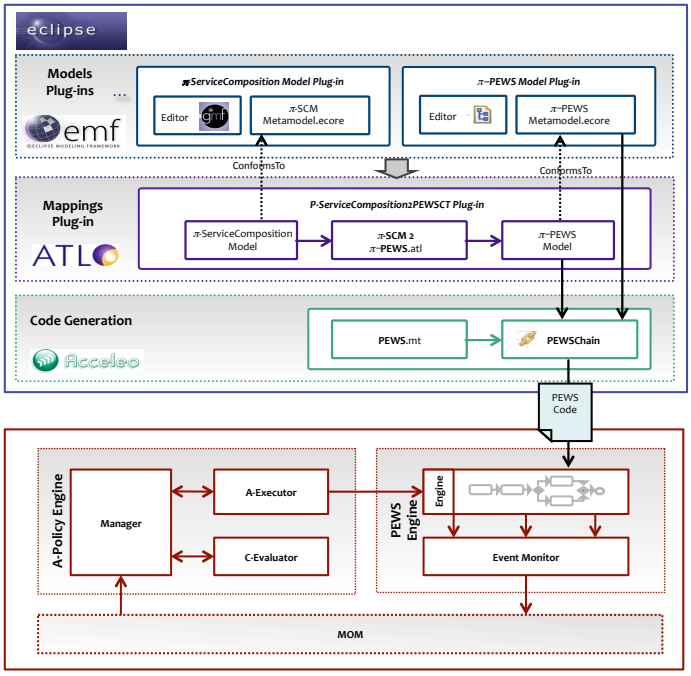

Figure 7: π-SOD-M Development Environment

- 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 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 7, 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

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

[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

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

8.2. 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.

9. 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.

9.1. 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.

9.2. 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[**?** **?** **?** **?**], 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* [**?**]. 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 [**?**] B-machines refinements that describe the services and their compositions. [**?**] 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.

9.3. 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.

10. 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.