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

Khalid Belhajjame[a], Valeria de Castro[b], Umberto Souza da Costa[c], Javier A. Espinosa-Oviedo[d], Martin A. Musicante[c], Plácido A. Souza Neto[e], Genoveva Vargas-Solar[f,d], José-Luis Zechinelli-Martini[d]

[a] *Université de Paris - Dauphine – Paris, France*
[b] *Universidad Rey Juan Carlos – Móstoles, Spain*
[c] *Universidade Federal do Rio Grande do Norte – Natal, Brazil*
[d] *Universidad de las Américas-Puebla, LAFMIA – Cholula, Mexico*
[e] *Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte – Natal, Brazil*
[f] *CNRS, LIG-LAFMIA, Saint Martin d'Hères, France*

Abstract

Specifying non-functional requirements (NFRs) is a complex task, being usually dealt with on the later phases of the software process. The late inclusion of NFRs in the development may compromise the quality of the deployed application. This paper presents πSOD-M, a methodology and associated tools that *(i)* allows the early specification of non-functional requirements in a principled way: users are abstracted away from low level details; *(ii)* embraces the MDA philosophy, thereby generating (instead of manually developing) models (code) whenever possible. The proposed solution has been utilized in the context of an industrial case study.

Keywords: MDA, Non-Functional Requirements, Service-based software process.

1. Introduction

In Service-Oriented Computing [34], pre-existing services are combined to build an application business logic. The selection of services is usually guided by the *functional* requirements of the application being developed [8, 16, 35][1]. An important challenge of service-oriented development is to ensure the alignment between the

[1] Functional properties of a computer system are characterized by the effect produced by the system when given a defined input.

functional requirements imposed by the business logic and the functions actually being developed.

Functional properties are not the only aspect in the software development process. Non-functional properties, such as data privacy, exception handling, atomicity and, data persistence, need to be addressed to fit in the application.

Even if service-oriented computing benefits from reuse, this reuse is usually guided only by functional requirements. Ideally, non-functional requirements should be considered in every phase of the software development. Yet, they are partially or rarely methodologically derived from the specification, being usually added once the code has been implemented. In consequence, the development process does not fully preserve the compliance and reuse expectations provided by the service oriented computing methods.

The literature stresses the need for methodologies and techniques for service oriented analysis and design [34]. Existing approaches argue that the convergence of model-driven software development, service orientation, and business processes improvement are key for developing accurate software [45]. Model Driven Development (MDD) for software systems is mainly characterized by the use of models as a product [39]. These models are successively refined from abstract specifications into actual computer programs.

Our work proposes the π-*Service Oriented Development Method* (πSOD-M) to support non-functional aspects of service-oriented applications, taking into account both functional and non-functional requirements from the early stages of software development. πSOD-M is aligned with the MDD guidelines and proposes models, practices and techniques for the development of service-based applications. The technique proposes the use of *models* to specify a software system at different levels of abstraction. Models are organized according to the guidelines for structuring the specifications of software systems of the Model Driven Architecture (MDA) [29] approach. The goals of πSOD-M are to:

(i) Improve the development process by providing an abstract view of the application and aiding to ensure the conformance to its specification.

(ii) Reduce the programming effort through the semi-automatic generation of models for the application, in order to produce concrete implementations from higher-level models.

This paper is organized as follows. Section 2 summarizes the general principles of existing works for addressing NFP and associating them to service compositions. Sections 3 and 4 introduce respectively the meta-models of the πSOD-M and transformation rules. Section 5 describes the experiment that we conducted for validating

our methodology and discusses lessons learned. Finally, Section 6 concludes the paper and gives research perspectives.

2. Related Works

While Functional Requirements establish *what* is computed by an application, Non-Functional Requirements (NFRs) are concerned with *how* the task is preformed. NFRs include aspects such as performance, authentication and quality constraints. These requirements are usually specified by conditions, called Non-functional Properties. Non-functional properties are also referred to as constraints, quality attributes, quality goals, quality of service requirements and non-behavioral requirements [11, 13, 12]. Most research on NFRs focusses on the evaluation of compliance by the software system as a whole. In the case of service-based applications, non-functional requirements are related to the application itself as well as to its component services.

In [6, 47] non-functional properties of web services are classified according to three points of view, namely, *service level*, *system level* and *business level*. In [46] the authors use the terms *non-functional attributes*, *composition model entity* and *model entity* to classify different concepts related to NFRs. The notion of non-functional attribute is used to describe NFRs of the abstract process model. In the lower level, the composition is annotated with non-functional attributes.

D'Ambrogio [14] uses the term *quality category* to group similar *quality characteristics*. *Quality dimensions* are used to quantify an individual characteristic. For instance, the quality category *performance* groups characteristics such as *latency* and *throughput*. The development process is based on MDA and the authors also present a WSDL extension for describing the QoS of web services. A catalog of *QoS characteristics* is provided for the web service domain, including properties such as *availability*, *reliability* and *access control*.

Schmeling et al. [38] present an approach and a toolkit for specifying and implementing web service compositions with support to several NFRs. The approach defines abstraction levels, where the terms *Non-Functional Concerns*, *Non-Functional Actions* and *Non-Functional Attributes* are used at each level (in decreasing order of abstraction). Non-functional concern is a general term used to describe non-functional requirements (such as *security*, *reliability* or *transactional behavior*). Each concern is refined into a set of non-functional attributes. Each non-functional attribute represents some behavior, to be refined into non-functional actions. An example of non-functional action is *encryption*, which provides the implementation of the non-functional attribute *confidentiality*, which is part of the *security* non-functional concern.

Pastrana et al. [36] use a traditional design-by-contract approach [28]. They use the term *contract* to describe non-functional requirements. Contracts may have pre-conditions, post-conditions and invariants. Each contract defines *assertions* associated with *quality properties*. Each service may have as many associated *contracts* as needed.

Chollet et al. [10] associate (non-functional) *quality properties* to (functional) activities. The authors present a security meta-model that takes into account web service compositions. The NFRs considered are *authentication*, *integrity* and *confidentiality*. Each NFR is associated with a service activity.

Ceri et al.[9] use the notions of *policy*, *rule*, *condition* and *action model* to specify NFRs. Agarwal et al. [1] associate *service policies* to services. Each service may also have *properties*, such as *security* and *reliability*. Ovaska et al. [33] use the terms *quality attribute*, *category*, *conceptual layer* and *importance* to organize and classify NFRs. Other authors do not define specific terms to refer to NFRs; they use terms such as *attribute* [48, 7, 24], *property* [21], *factor* [30, 23], *characteristic* [18], *quality level* [17], and *value* [43, 7].

Despite of the different notations found in the literature for classifying NFRs, some non-functional requirements are frequently considered, such as *security*, *performance*, *reliability*, *usability*, and *availability*. However, distinct hierarchies and models are proposed for NFRs, according to different points of view. We have identified a number of approaches [14, 10, 38, 7, 21, 33] that use MDD (Model Driven Development) for designing and developing applications.

The authors in [43, 48] use formal methods to define a service-based development processes that take NFRs into account. In [1, 36] ontologies are used to define and model NFRs, whereas in [46, 23] Business Process Modeling is used for system specification, including NFRs. In the method defined in [46], each task and data item of the application can be annotated with functional and non-functional attributes (NFAs). Functional and non-functional attributes are independently defined. They are attached to specific tasks later in the development of the application. NFAs for data considers *value* and *range*, whereas NFAs for tasks include *cost*, *time*, *resources* and *expressions*.

The proposal in [43] presents steps for selecting services by taking QoS information into account. The proposed steps are: *(i)* identification of relevant QoS information; *(ii)* identification of basic composition patterns and QoS aggregation rules for these patterns; and *(iii)* definition of a selection mechanism of services. The QoS properties considered are *performance*, *cost*, *reliability* and *availability*.

Karunamurthy et al. [25] use the term *non-function parameters* to define NFRs, such as *cost*, *response time*, *availability*, *security*, *reliability* and *reputation*. The *Non-*

Functional Specification Language (NFSL) is proposed as a domain specific language (DSL) to express *non-function parameters*.

Liu et al. [27] use the term *QoS parameter* to describe non-functional requirements such as *cost*, *execution duration*, *accuracy*, *security*, *integrity*, *availability* and *reliability*. In the same way, Tran et al. [44] use the term *QoS policies* to classify similar non-functional requirements.

Li et al. [26] associate *dimensions* to *QoS parameters* to classify NFRs. For instance, the *time* dimension is associated to the *execution time* and *communication time* parameters; the *spatial* dimension is associated to the *storage capacity* and *message length* parameters; the *reliability* dimension is associated to the *availability* and *reliability* parameters and the *cost* dimension is associated to the *service cost* parameter. Rumpel et al. [37] associate *quality requirements* to *quality properties*. Quality requirements are intended to be specified as constraints.

Most works agree on distinguishing three points of view, namely the point of view of the organization (*Business* view), of the individual service providers (*Service* view) and of the composition designer (*System* view). The Business view is concerned with the business logic (*i.e.*, an abstract level of tasks, defined by the guidelines and constraints imposed by the organization). Service and System views guides the implementation of the software solution: The Service level is related to the building blocks of the application. It may use web services provided by third party sources. The System level deals with the coordination of services, to implement the business logic. Our proposal, shown in the next section, follows these ideas.

3. Modeling reliable service compositions with πSOD-M

This section presents πSOD-M, an MDD-based methodology for building service compositions with non-functional requirements. πSOD-M provides meta-models for modeling functional and non-functional requirements organized in three levels (Figure 1): CIM (*Computational Independent Models*), PIM (*Platform Independent Models*) and PSM (*Platform Specific Models*). Given high-level models specified at the CIM level, πSOD-M proposes the semi-automatic refinement of these models, for the generation of a set of models at the other levels of abstraction. The refinement process is driven by transformation rules specified between the meta-models.

The next subsections present the meta-models from higher to lower levels of abstraction. The presentation is exemplified by a running example. The transformations between models are presented in section 4.

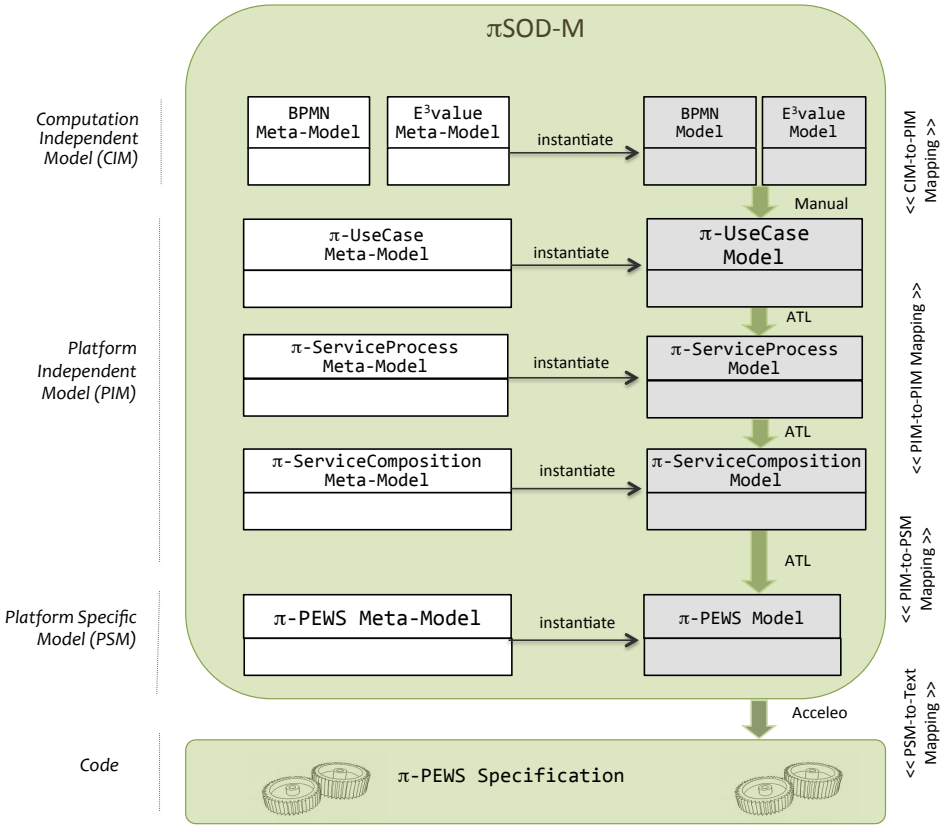

Figure 1: πSOD-M Overview.

3.1. Computation Independent Models

This level focuses on the highest-level view of the system, including its business and requirement specifications. At this stage of the development, the structure and system processing details are still unknown or undetermined. πSOD-M uses the *e*^{3}*value* [22] and *BPMN* [31] meta-models for this purpose.

3.1.1. E^3*value meta-model*

The e^{3}value model of an application identifies the value/information exchange between components of the system. It is a business model that represents a business case graphically as a set of value exchanges[2] (∇ $\triangle$) and value activities (rounded

[2]See Figure 2.

boxes) performed by business actors (squared boxes). The model is well suited to enhance the understanding of the environment in which an application is being developed. It defines *dependency paths*, showing the value exchange between providers and end users when they ask for a service. A dependency path has a direction and consists of a sequence of linked dependency nodes. It starts with a *start stimulus* node and ends with an *end stimulus* node (see Figure 2). Dependency paths may also contain *OR* and *AND* elements (both for initiate and join alternative and parallel paths).

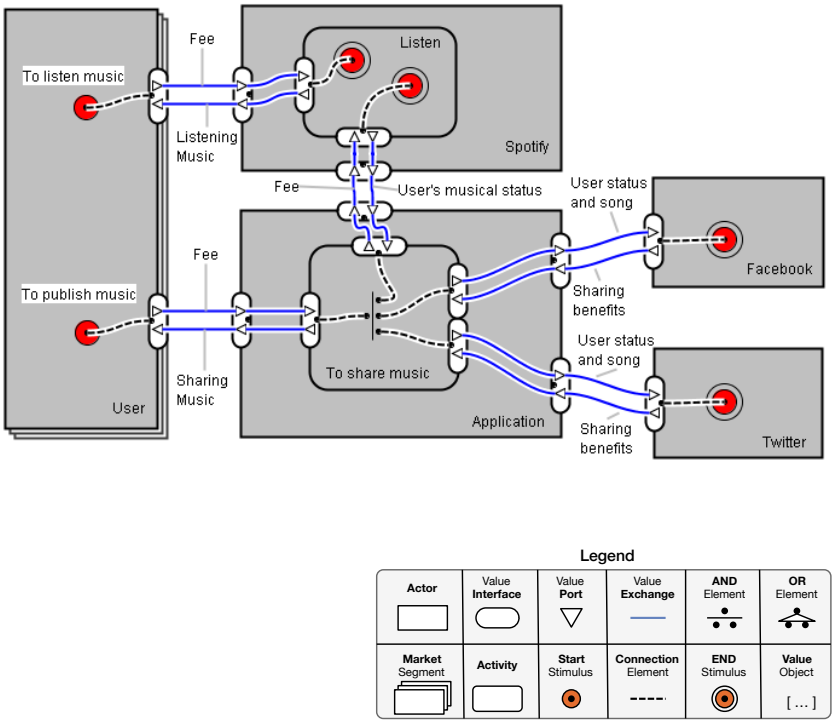

Figure 2: e3value model for "To Publish Music".

Example 1 (To Publish Music) Let us consider the scenario "To Publish Music", used as a running example in this section: An organization wants to provide the service based application "To Publish Music" that monitors the music listened by a user during some periods of time and sends the song title to this person's Twitter and Facebook accounts. In this way, the user will have her status synchronized in Twitter and Facebook (i.e., either the same title is published in both accounts or it

is not updated) with the title of the music she is listening in Spotify. The application is based on three external actors (*Spotify, Twitter* and *Facebook*). The following (external) services will be used by the application:

- The music service Spotify exports a method for obtaining information about the music a given user is listening:
  - get-Last-Song (userid): String ;
- Facebook and Twitter services export methods for updating the status of a given user:
  - update-Status (userid, new-status): String;

The e^{3}value model for "To Publish Music" is shown in Figure 2. The e^{3}value value model shows Spotify and a private application (which is also a service) that directly interact with users for providing free services for listening and publishing information about music being listened by users. The private application interacts with Spotify for obtaining free information about the flow of music being listened by a user in return of a fee (i.e., premium subscription). Finally, the private application interacts with Facebook and Twitter for updating the user's status. We can see this interaction as non material benefit sharing (as users subscribe to their networks and are active on them thanks to the private application). □

The e^{3}value model is used to model the (economic) value exchange among the actors involved in an application. It is also necessary to understand the business process of the application and the conditions in which the different steps of this process are executed input/out data and the dependencies among these steps. Therefore, our methodology proposes the BPMN meta-model as a tool for modeling this aspect of the application.

3.1.2. BPMN meta-model

BPMN [31] is a graphical representation that establishes the business process of the application through a high-level workflow. The next example illustrates the use of this meta-model.

Example 2 (To Publish Music *(cont)*) Figure 3 shows the BPMN model[3] of the scenario. It starts by contacting the music service Spotify for retrieving the user's

[3]Details on BPMN (Business Process Management Notation) can be found in http://www.bpmn.org/

musical status (activity Get Song). Twitter and Facebook services are then contacted in parallel for updating the user's status with the corresponding song title (activities Update Twitter and Update Facebook). □

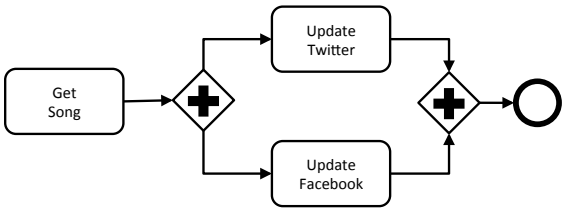

Figure 3: BPMN model for "To Publish Music".

The CIM level models are the basis for the development of the application. The information presented at this level is (manually) refined into PIM-level models of the methodology πSOD-M described next. Notice that, at this level, the models are (informally) used to describe both functional and non-functional requirements.

3.2. Platform Independent Models

This level focuses 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. Our methodology defines three PIM-level meta-models: *π-UseCase*, *π-ServiceProcess* and *π-ServiceComposition*.

3.2.1. π-UseCase meta-model

Figure 4 presents the *π-UseCase* meta-model. The goal of *π-UseCase* is to be the first representation of the application in terms of functionality, as well as to represent its non-functional requirements. The notion of *policy* is used to describe NFRs. (In Figure 4, we have highlighted the concepts related to NFRs). The *π-UseCase* meta-model extends the UML Use Case meta-model for describing non-functional requirements with the concepts: Business service, End consumer, Requirement, Use Case, Composite Use Case, non-functional requirement, non-functional attribute and Constraint. An End Consumer is represented by an Actor. An Actor is related to Use Cases (from the original UML definition), while a Composite Use Case is a set of actions performed by the system which can be broken into different Use Cases. Business Service aggregates several Use Cases, a service can be expressed by one or more use cases. The Business Collaborator concept is represented through Packages. A Business Collaborator represents an external service or a system that interacts with the

application that are being modeled. Each Business Collaborator combines the features described in each Package. Thus, the services functions can be specified as being grouped into packages.

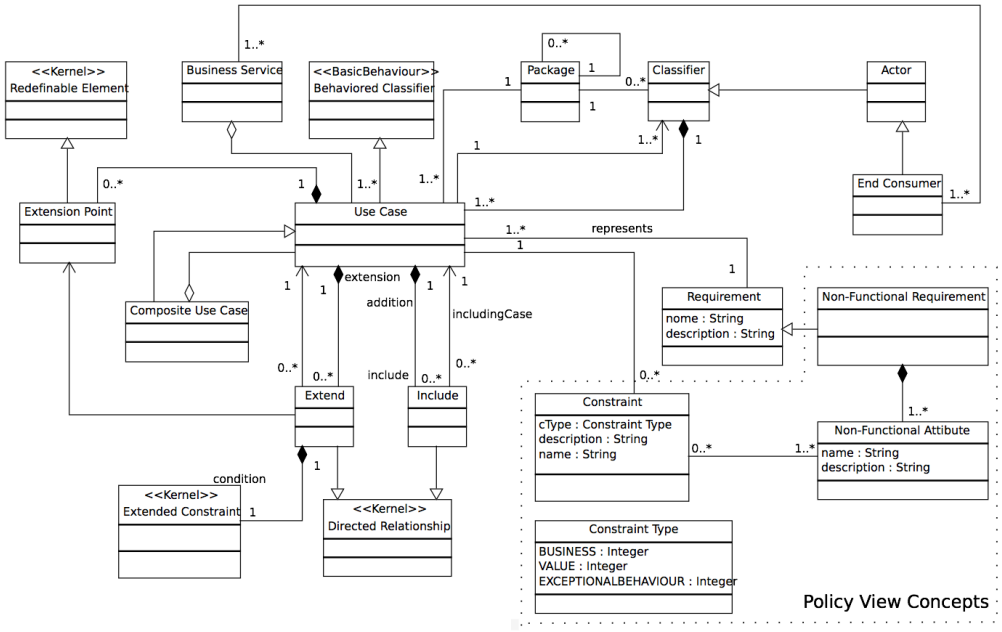

Figure 4: π-UseCase Meta-Model.

The Non-functional requirement and Non-functional Attribute concepts are represented as part of description of Use Cases and Constraints. A Use Case may have several Constraints. Each Constraint has its name, description, and which ones should be checked during the execution of the application. Each Constraint is represented as a stereotyped (constraint) use case. There are three kinds of constraint: *(i)* A restriction may be associated to data (Value Constraint), represented as the stereotype "value"; *(ii)* A restriction may correspond to business rules (Business Constraint), represented as the stereotype "business"; and *(iii)* A restriction may describe Exceptional Behavior constraints, represented as the stereotype "exceptional_behavior". These concepts are present in the following example.

Example 3 (To Publish Music *(cont)*) Figure 5 shows a π-UseCase model for our example application. We consider that, besides the service composition for implementing the application, it is necessary to model other requirements that represent

the (i) conditions imposed by services usage –for example, the fact that both Facebook and Twitter require authentication protocol in order to call their methods for updating the wall; (ii) the conditions stemming from the business rules of the application logic, (e.g., the fact that the walls in Facebook and Twitter must show the same song title and if this is not possible then none of them is updated).

The "To Publish Music" Business Service expects the Facebook or Twitter user status to be changed every time a user starts listening a new song. Therefore, it is necessary to perform a social network authentication with the users data. Each social network uses different services and different forms of authentication. The authentication constraint is required to update a music status. The restriction is stereotyped as a value constraint, because the users id and password are verified. Figure 5 shows the buy music, download music, listen music and pay use cases. The process of buying a song requires the user private data for a Spotify account, and also a secure connection, represented as value and business constraint, respectively. For payment, the user must provide the data of payment card or PayPal account login and password, represented as value stereotype. Another restriction is that the minimum payment value is 2 euros. □

3.2.2. π-ServiceProcess meta-model

The *π-ServiceProcess* meta-model extends the UML activity diagram with the concept of *contract* to represent constraints over data and actions. This concept is used to model groups of constraints described in the *π-UseCase*. As shown in Figure 6, the concepts of the *π-ServiceProcess* meta-model are: Contract, Assertion, Exceptional behavior, Activity, Service Activity, Action and Constraint. The portion of the figure enclosed by a dotted line defines those concepts related to the representation of NFRs.

A specific *π-ServiceProcess* model shows the set of logically related activities to be performed in a service-based application. So, the activities of this model represent a behavior that is part of the business logic of the application. A *π-ServiceProcess* model contains three main elements: (i) service process, (ii) service activity and (iii) activity contract. A service activity represents an operation that is part of the execution flow, and it is modeled as an Action. An activity contract represents the Non-functional Requirement that is also part of the execution flow of a service, identified as a stereotyped activity (assertion). The Assertions associated to an Action compose a Contract. Using the concepts Contract and Assertion it is possible to specify each activity service, by defining its pre-conditions and post-conditions.

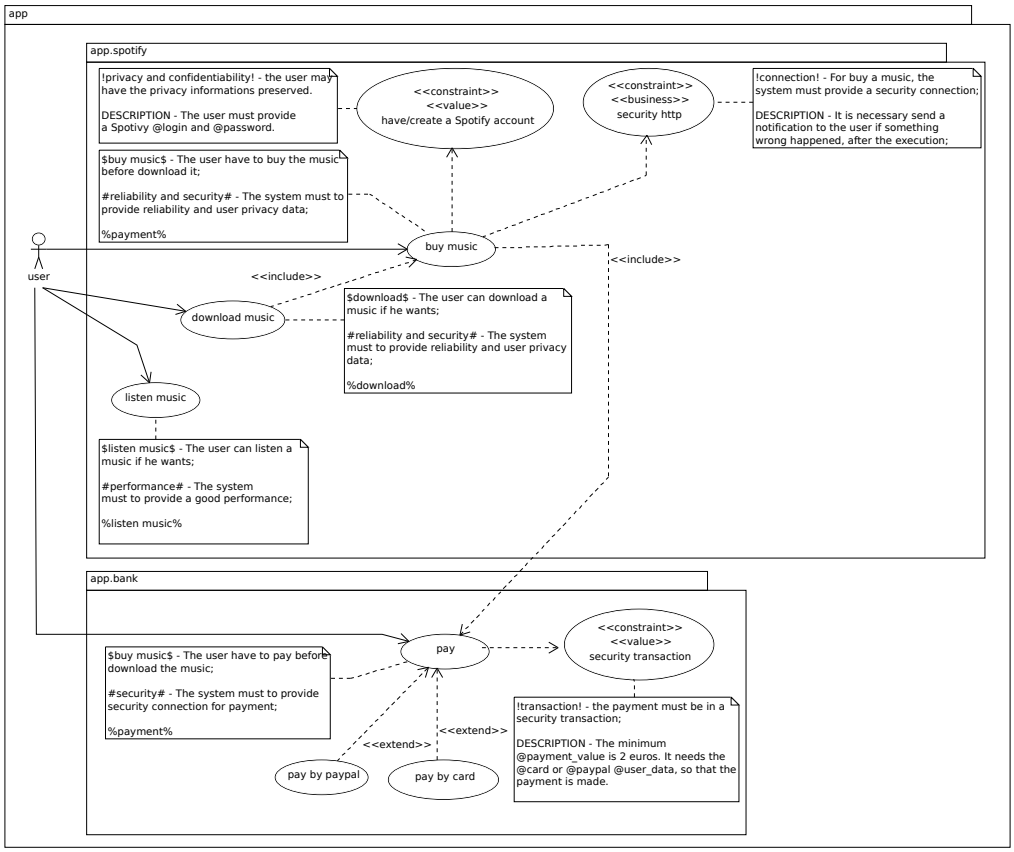

Figure 5: π-UseCase model for "To Publish Music".

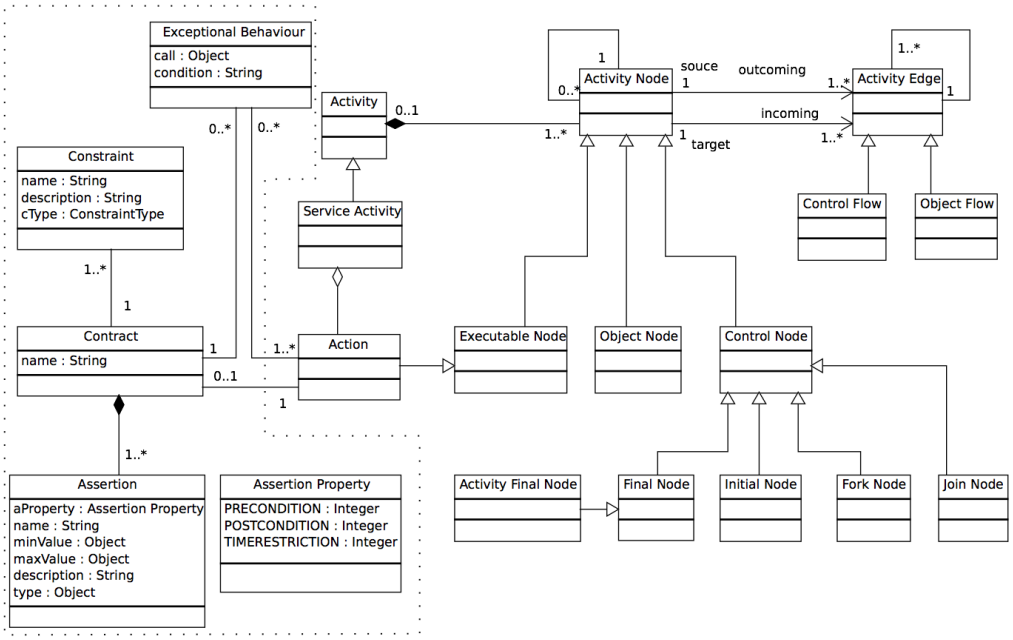

Figure 6: π-Service Process Meta-Model.

Example 4 (To Publish Music *(cont)*) Considering the example scenario, the contract based process of activities is shown in figure 7. The buy music and publish music services (update Twitter and Facebook) have pre- and post-conditions assertions that are composed into a contract for each service. The buy music pre-conditions consist in verifying: (i) if the User data are correct; (ii) if the User is already logged in Spotify; (iii) if bank account information are correct and; (iv) if there are enough funds in the bank account to cover the payment. A post-condition ensures the complete transaction and verifies if a notification was sent to the user and Spotify, about the payment authorization. There are four assertions for the buy music action, and each assertion has been detailed with the assertion property and predicate that must be verified. To update services, depending of each service, there may be different restrictions. As an example, a new verification of user data and message format is appropriate (maximum 140 characters), in the case of Twitter. In the case of Facebook, it is required that the user is already logged in Spotify and these data are the same as Facebook. As post-condition, the application ensures that the Facebook service sends a notification of success. To update Twitter a pre-condition is required, while to update Facebook it is necessary to check a pre-condition and a confirmation notice (modeled as post-condition). As a pre-condition for "twitter

update" it is necessary that (i) the music format is correct and (ii) the twitter login and password are correct for the update. □

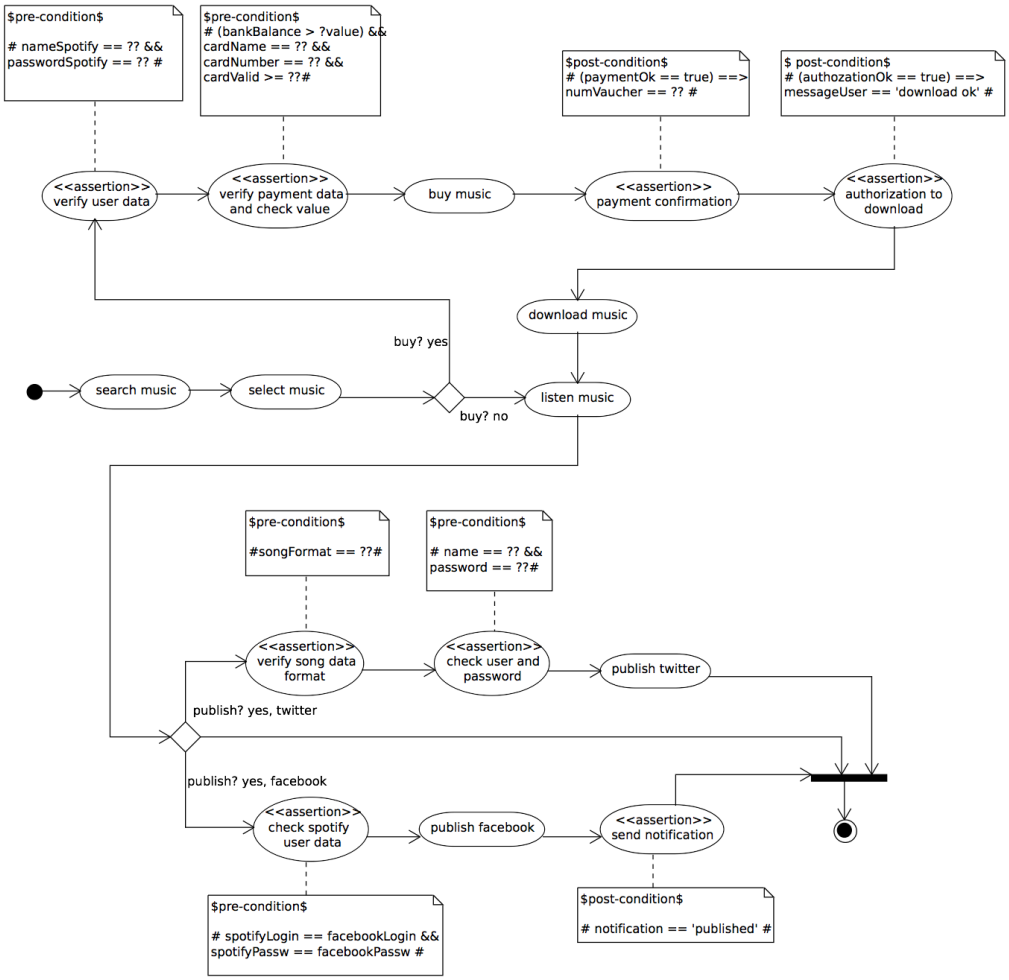

Figure 7: π-ServiceProcess model for "To Publish Music".

3.2.3. π-ServiceComposition meta-model

As shown in Figure 8, the π-ServiceComposition meta-model provides meta-classes to represent workflows[4] that model business processes. The *π-ServiceCom-*

[4]Workflows are transformed into implemented service compositions.

position meta-model extends the UML activity meta-model with the concept of *A-Policy* to group contracts with similar non-functional requirements. For instance, security and privacy restrictions may be grouped into a security policy. The (π-ServiceComposition) meta-model defines:

- A BUSINESS COLLABORATOR meta-class, to represent the classes of entities that collaborate in business processes by performing some required action. An instance of this meta-class is graphically represented as a partition in the activity diagram. A collaborator can be either internal or external to the system. When the collaborator of the business is external to the system, the attribute IsExternal[5] of the collaborator is set to **true**.

- ACTIONS, a kind of EXECUTABLENODE, are represented in the model as a class activity instance of the meta-class ACTION. A class action represents some type of transformation or processing. There are two types of actions: i) a WebService (attribute Type is WS); and ii) a simple operation called an ACTIVITYOPERATION (attribute Type is AOP).

- The SERVICEACTIVITY meta-class represents classes of composite activity types that must be carried out as part of a business service and is composed by one or more executable nodes.

- In order to represent constraint types associated to services compositions, we introduced the meta-classes RULE and A-POLICY (see blue meta-classes in the π-ServiceComposition meta-model in Figure 8). We model non-functional constraints by using the notion of *A-policy* [19, 20]. An *A-policy* is defined by attributes and rules. The conditions of each rule are evaluated and, if they are not verified, the actions of the rule will be performed. The RULE meta-class represents the types of event-condition-action rules where the EVENT part represents the moment in which a constraint is evaluated. An *A-policy* defines variables and operations that can be shared by the rules and that can be used for expressing their Event and Condition parts.

Instances of this meta-model are represented as UML activity diagrams.

Example 5 (To Publish Music *(cont)*) To illustrate the use of the π-Service Composition meta-model, we define a model for the "To Publish Music" scenario (Figure 9). We model a business process that consists of three service activities:

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

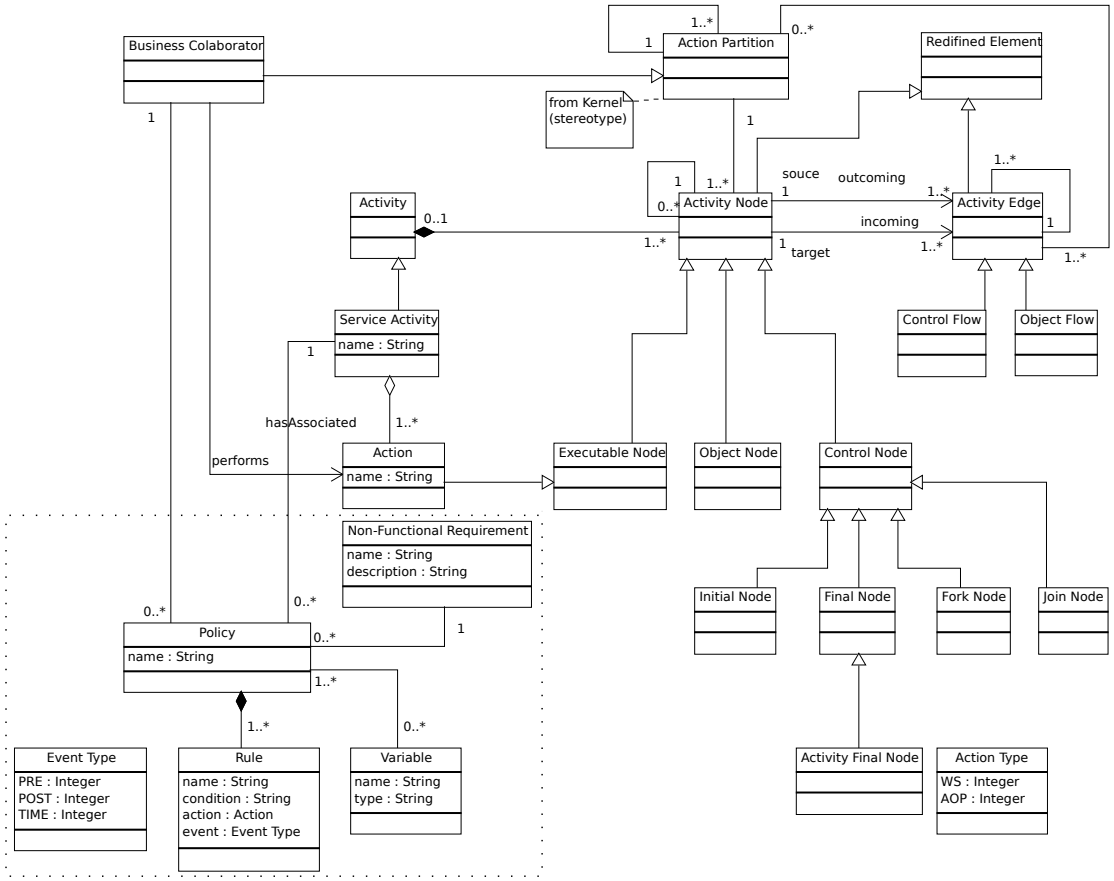

Figure 8: π-Service Composition Meta-model.

Listen Music, *Publish Music* and *Confirmation*. The *Publish Music* activity calls the *Facebook* and *Twitter* services. Both *Facebook* and *Twitter* require authentication. Two authentication policies are required, one for *Twitter* and another for *Facebook*. In this model, there are three external business collaborators (*Spotify, Twitter* and *Facebook*). The model also shows the business process of the application that consists of three service activities: *Listen Music*, *Publish Music* and *Confirmation*. Note that the activity *Publish Music* calls the actions of two service collaborators namely *Facebook* and *Twitter*. Both *Facebook* and *Twitter* services require authentication protocols in order to execute methods that read and update the user data. In the example, we associate two authentication policies, one for the open authentication protocol, represented by the class OAuthPolicy at *Twitter*, associated to the activity

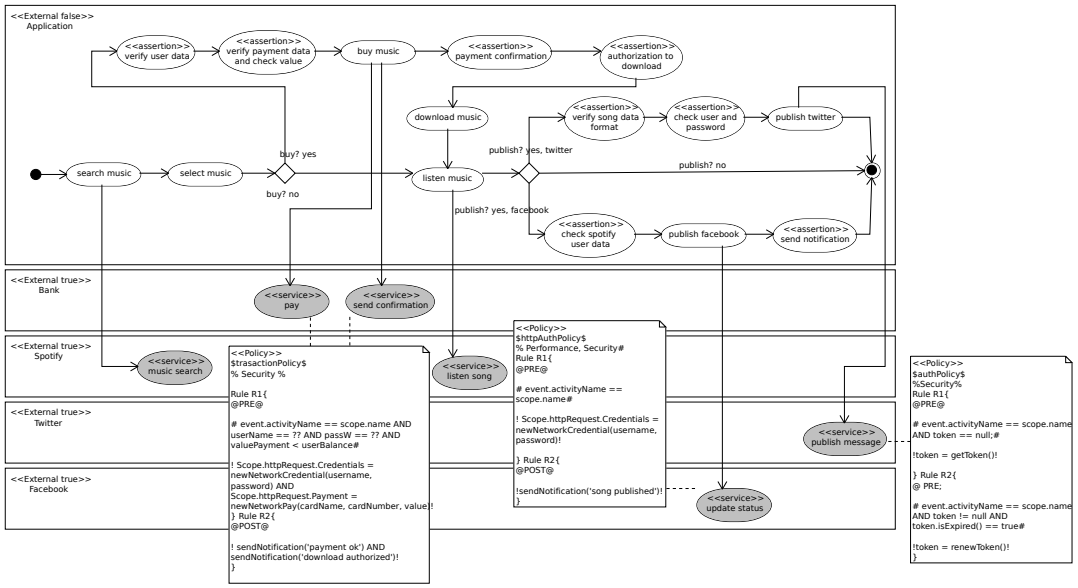

Figure 9: π-ServiceComposition Model for the "To publish music" business service.

UpdateTwitter (see Figure 9). In the same way, the *Facebook* class HTTPAuthPolicy, for the http authentication protocol is associated to the activity UpdateFacebook. OAuthPolicy implements the open authentication protocol. The *A-policy* OAuthPolicy has a variable Token, used to store the authentication token provided by the service. This variable is imported through the library OAuthPolicy.Token. The A-policy OAuthPolicy defines two rules, both can be triggered by events of type ActivityPrepared: (R_1): If no token has been associated to the variable token, then a token is obtained ; and (R_2): if the token has expired, then it is renewed. Notice that the code in the actions profits from the imported OAuthPolicy.Token for transparently obtaining or renewing a token from a third party. HTTPAuthPolicy implements the HTTP-Auth protocol. 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. □

Once the π-Service Composition Model has been defined, then it can be transformed into a lower level model (in our case, π-PEWS) that gives support to code generation. The π-PEWS meta-model is described in the next section.

3.3. Platform Specific Models

This level focuses 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. At the PSM level we have lower-level models that can be automatically translated into actual computer programs. We have defined one meta-model at this level.

The *π-PEWS* meta-model provides concepts for modeling service compositions. Instances of this meta-model are textual descriptions of service compositions that can be translated into any service composition language, such as BPEL [3] or PEWS [4, 42].

PEWS [5, 42] is a notation to express service compositions. The language is based on the notion of Path Expressions [2] and can easily be translated into any actual composition language, such as BPEL [3]. Figure 10 presents the π-PEWS meta-model, where we identify classes to describe:

- Service compositions: NAMESPACE representing the interface exported by a service, OPERATION that represents a call to a service method, COMPOSITEOPERATION, OPERATOR and PATH for representing service compositions. A PATH can be an OPERATION or a COMPOUND OPERATION. A COMPOUND OPERATION is defined using an OPERATOR. The language defines operators to denote guarded operations ($[C]S$); sequential (.), parallel ($\|$) and alternative (+) compositions; as well as sequential ($*$) repetition.

- *A-Policies* that can be associated to a service compositions: A-POLICY, RULE, EVENT, CONDITION, ACTION, STATE, and SCOPE. A brief description of these classes is given next.

Figure 10 shows that each A-POLICY is associated 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) or a PATH. Each A-POLICY groups a set of ECA rules with a classic semantics, i.e, *when an event of type E occurs, if condition C is verified then execute the action A*. In this way, an *A-policy* represents a set of reactions to be possibly executed when one or several events are notified.

Example 6 (To Publish Music *(cont)*) Figure 11 shows the π-PEWS code resulting from the π-service composition model of our scenario example. Notice that the code contains namespaces and definitions (obtained by using information about the business collaborators of the π-SCM model), a workflow expression *(Path)* containing operation calls and contracts (derived from the *A-Policies*). □

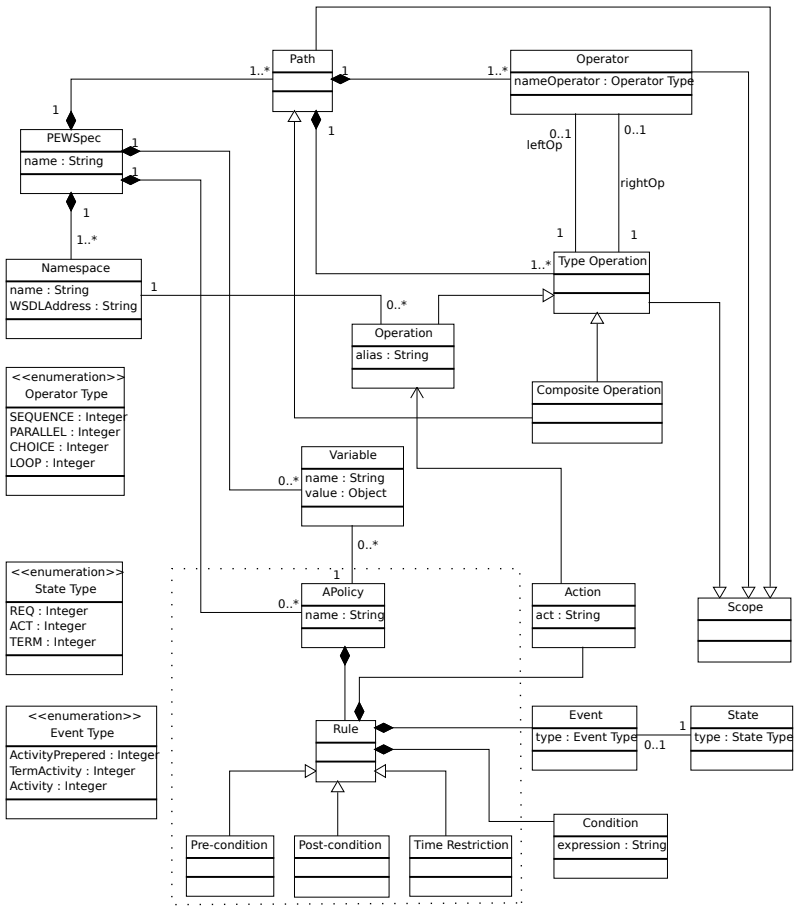

Figure 10: π-PEWS Meta-model.

4. Transformation rules

πSOD-M proposes intra-level transformation rules between π-use case to π-service process models, π-service process to π-service composition models, as well as inter-level transformation rules between π-service composition to π-PEWS models. These rules are used to build a semi-automatic tool (the πSOD-M environment plugin for Eclipse). The transformations from CIM to PIM level models is not automatized, due to the informal nature of CIM level representations.

4.1. From π-UseCase to the π-ServiceProcess

The refinement of a (composite) π-Use Case model into a π-Service Process model is driven by the principle of expressing a set of π-use cases (i.e., use cases with

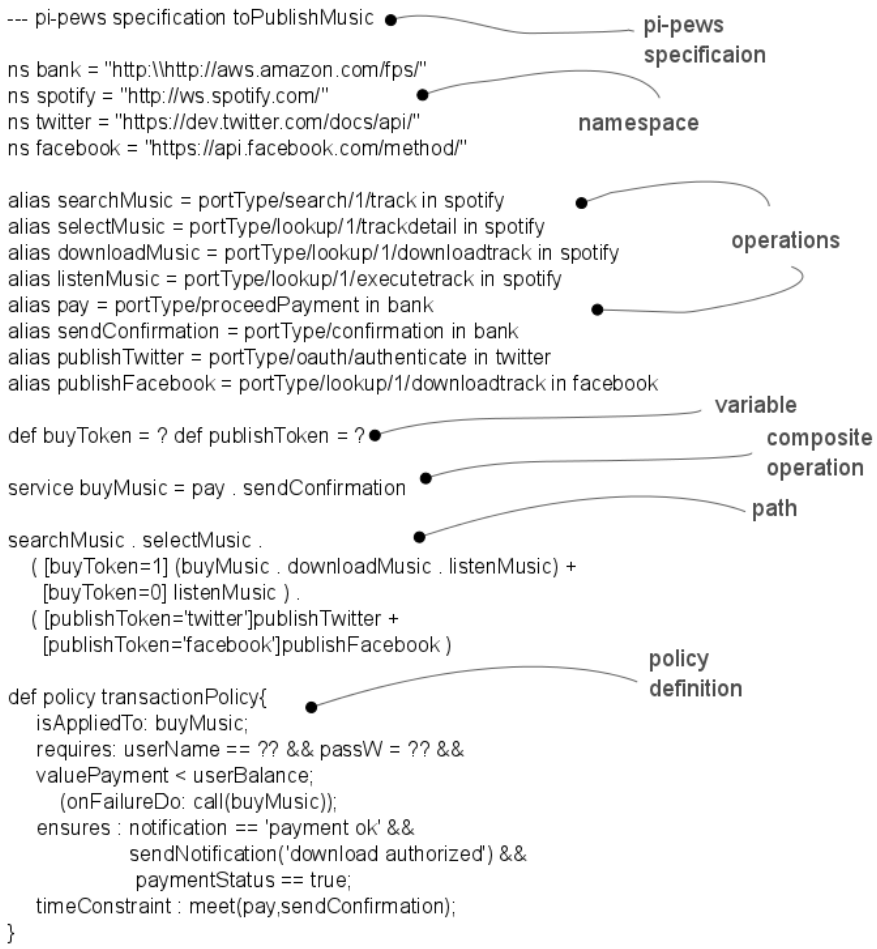

Figure 11: π-PEWS Specific and Policy Representation.

constraints) in terms of a business process. The resulting model consists of actions related by a control flow and contracts specifying NFPs.

As defined by the π-ServiceProcess meta-model (Figure 6) an ACTIVITY SERVICE consists of a composition of entities of type ACTION. Thus, every π-use case is transformed into an Action of the target π-Service Process model. Every Extend relationship identified in a π-Use Case model is transformed into a Fork node [15, 40]. If the Extend relationship concerns just one use case, it is transformed into an Action inside one flow of the fork node. Otherwise, several use cases are transformed into different Actions that belong to different flows departing from the fork node.

A Constraint associated to a Use Case is transformed into an Assertion. The set of resulting assertions are grouped into a Contract. Constraints are transformed according to their type: BUSINESS constraints and VALUE constraints with the isExceptionalBehaviour attribute set to false are transformed into Assertions; VALUE constraints with the isExceptionalBehaviour attribute set to true are transformed into Exceptional behaviors.

In order to transform constraints of type Value Constraint, the designer must specify thresholds to be associated to the assertions of a contract. By default, value constraints are transformed into pre-conditions and business constraints are transformed into post-conditions.

The relationships of type EXTEND and INCLUDE determine the way the business process is expressed as a workflow. The generated workflow is composed by Fork and Join nodes, CONTROL FLOW constructors, as well as entities of type ACTION.

Include use case entities are transformed into an Action sequence. A USE CASE element is transformed into an Action. A set of n USE CASES is transformed into an $n-1$ Object flow elements.

The details of these transformations are not included here (due to space restrictions). They can be found in [41].

Example 7 (To Publish Music *(cont)*) The rules presented above have been applied to the model in Figure 14, in order to obtain the π-Service Process model for our running example (Figure 7).

The "listen music" use case is transformed into a Service Action. This Action represents a Spotify service function that can be invoked to play the music. For the "publish music" use case, constraints are transformed into a set of assertions that are grouped into a Contract ("publishMusicContract") associated to the Action "publishMusic". The "download music" use case includes the payment process to buy the music. Thus, these use cases are transformed into Actions, and a Service Activity that aggregates these Actions. This Service Activity is transformed into a sequence

flow on the π-service process model. The same rule is applied to the "publish music" use case, which has two extended use cases, to publish on Twitter and Facebook. □

4.2. From π-ServiceProcess to π-ServiceComposition

The principle of the transformation of a π-Service Process model into a π-Service Composition model is to group Contracts into A-Policies and Actions into Service Activities.

Each Assertion of a Contract is transformed into a Rule. Rules concerning the same NFP are grouped into A-Policies. Each Assertion of a Contract is transformed into a Rule:Condition attribute. If the Assertion has a value type, the name and the attributes are transformed into Variables in the target model. The Assertion: aProperty attribute can have different transformations, according to: *(i)* a Precondition is transformed into Pre; *(ii)* a Post-Condition is transformed into a Post; *(iii)* a TimeRestriction is transformed into Time.

Some elements of the π-service composition model are obtained from the π-UseCase model: A Package in the π-use case model is transformed into a Business Collaborator; Non-Functional Attributes of the π-use case model are grouped into Non-Functional requirements of the π-ServiceComposition model. These requirements are associated to a Policy (from the π-ServiceProcess model).

Actions and Service activity of a π-Service Process model are transformed into their homonym concepts of the π-Service Composition model.

Finally, Actions are grouped into a Business collaborator.

Example 8 (To Publish Music *(cont)*) Considering the example scenario, the model in Figure 16 was obtained by applying the rules above to the model depicted by Figure 7. The "securityLoginPolicy" consists of a set of Rules that were transformed from the Assertions in π-service process model. The information about Facebook and Spotify (both of them Business Collaborators) come from entity of type PACKAGE in the π-use case model. □

4.3. From π-ServiceComposition to π-PEWS

This section describes the PIM to PSM transformations from a π-Service composition model to a π-PEWS model. We distinguish two groups of rules: *(i)* those transforming service composition entities into workflows; and *(ii)* those that transform A-Policies into Contracts.

Single actions (represented by Action and Action:name) are transformed into individual service Operations. Complex actions (represented by ServiceActivity and ServiceActivity:name) are transformed into (named) composite operations that defines a (named) workflow of the application. Composition patterns expressed using

the operators *merge, decision, fork and join* are transformed into their corresponding workflows of the π-PEWS model.

The A-Policies defined for the entities of a π-service composition model are transformed into A-policy entities, named according to the names expressed in the source model. The transformation of the rules expressed in a π-service composition is guided by the event types associated to these rules. The variables associated to an A-Policy expressed in a π-service composition model as <Variable:name, Variable:type> are transformed into entities Variable with attributes Name and Type directly specified from the elements Variable:name and Variable:type of a π-service composition model.

Events of type Pre, Post and Time generate, respectively, Pre-conditions, Post-condition and TimeRestrictions.

Example 9 (To Publish Music *(cont)*) Figure 1 shows the π-PEWS code resulting from the π- service composition model of our scenario example. □

4.4. Implementation

We have developed tools for aiding the user to define and transform the models for all the levels, except for the CIM into PIM level, which should be manually performed.

Our tool is implemented as a series of Eclipse plug-ins:

- We used the Eclipse Modeling Framework (EMF)[6] for implementing the π-Service Composition and π-PEWS meta-models. Then, from these meta-models, we developed plug-ins to support their graphical representation.
- We used ATL[7] for implementing the mappings between models.
- We used Acceleo[8] for implementing the code generation plug-in for generating executable code. It takes a π-PEWS model and generates the code to be executed by the *A-Policy* based service composition execution environment.

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

[7]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.

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

5. Applying πSOD-M: The *FlyingPig* use case

We validated our methodology by developing a use-case concerning risk assessment for financial companies as implemented by the ORCA System[9]. Risk assessment is implemented by an interactive business process based on the exchange of a series of questionnaires intended to evaluate the risks implied by the client business practice. Examples of business practices are: the conditions and protocols used to perform confidential transactions, the physical security for accessing reserved areas (such as computing server installations). The information gathered by the questionnaires is used to evaluate whether there are risky practices within the business processes of the company, as well as to propose amendments to these practices. The ultimate goal of the risk assessment is to determine a degree of compliance to existing standards. By analyzing the questionnaires, ORCA detects risky practices, proposes solutions and triggers further assessment processes to ensure that the solutions have been implemented.

Our goal is to model a service based application (called *FlyingPig*), for providing risk assessment as a service. In order to provide this functionality, *FlyingPig* would benefit from ORCA's legacy services providing storage, assessment and data visualization functions.

In the following sections, we describe the results of applying πSOD-M to develop the *FlyingPig* risk assessment system. The models presented next were generated as a result of interacting with software developers at GCP Global.

5.1. Computation-Independent Models (CIM)

Figure 12 shows the value model for the *FlyingPig* application. It is a business model that graphically represents a business case as a set of value exchanges (▷ and ◁) and value activities (rounded boxes) performed by business actors (squared boxes). We identify two business actors: *ORCA* and *Broker*. Brokers emit requests for risk assessment for one or more companies. ORCA has two value activities which are services that provide an economical benefit: *Identify Amendments* and *Assess Risk Situation*. The values exchanged between ORCA and the brokers are: (i) *Questionnaire and Evidences* filled with information about the client company; (ii) *Amendments* which are ORCA's recommendations, based on the data provided by the answers to questionnaires; (iii) *Evaluation Reports* for the client companies; and (iv) the risk assessment *fee*.

[9]The ORCA System is a trademark of GCP Global (www.gcpglobal.com).

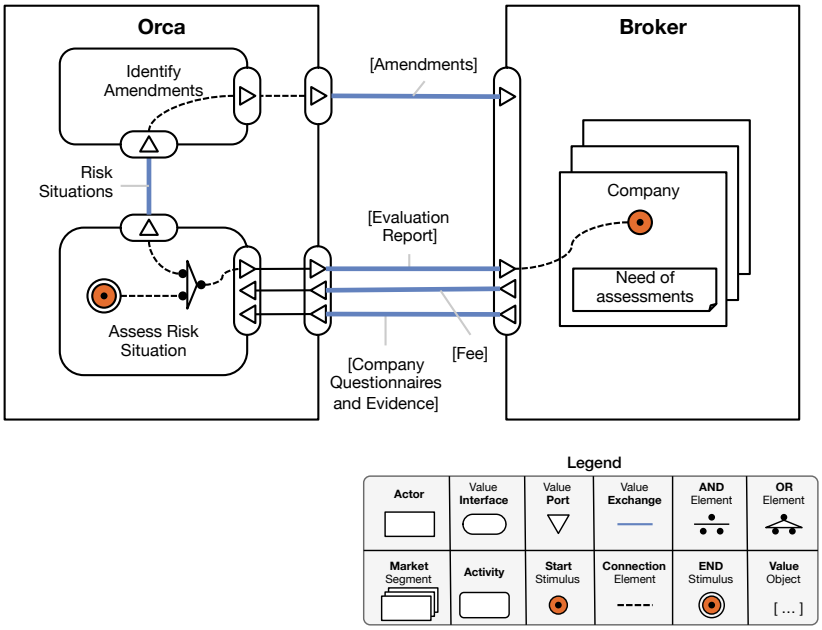

Figure 12: E3value model for *FlyingPig*.

The dependency path in Figure 12 initiates with the need of assessment emitted by a particular company. Once this need has been declared, the value exchanges between ORCA and Broker are triggered. The client company provides ORCA with information (answers to a questionnaire), evidence (to support the information) and a fee (monetary value). Then, ORCA suggests amendments (recommendations to change practices) and provides an evaluation report.

Figure 13 shows the BPMN model for the *FlyingPig* scenario. The BPMN model for *FlyingPig* is partitioned for better understanding the process in which the value exchanges occur. The model includes two pools representing the *ORCA* system and the *Brokers*. Brokers have two lanes, the client *Company* and a *User*. The user is a contact member of the company, who coordinates the assessment process. This process involves other members of the company as well.

The risk assessment process starts after a request from a company. This corresponds to the value model, in which the start stimulus triggers the whole process. The request leads to the definition of a group of users that will answer questionnaires for evaluating risk. Questionnaires are considered tasks to be performed by users. Other tasks include amending a "risky situation" as well as producing evidence to

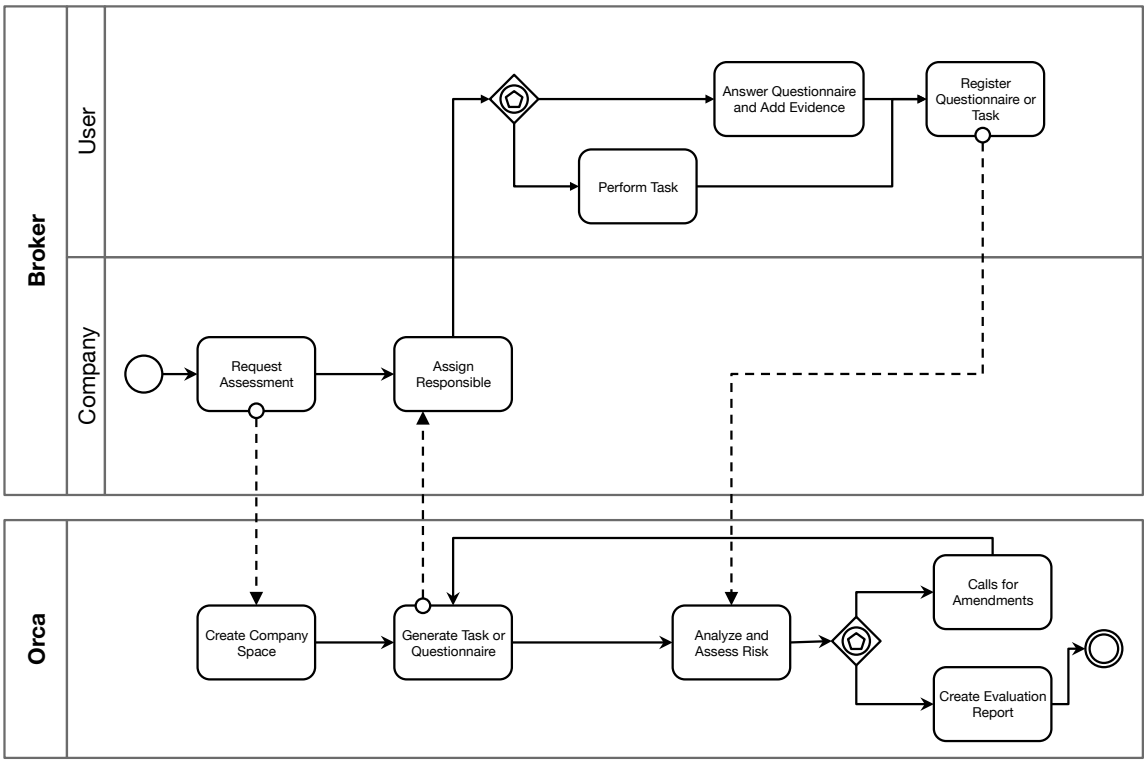

Figure 13: BPMN model for *FlyingPig*.

show that a specific risk has been eliminated[10].

Once tasks have been completed, they are stored and analyzed to generate a list of non-compliant situations, associated to their corresponding *calls for amendment* (if needed) or a report specifying a compliance level, incidents and a risk map. During the process of analyzing a questionnaire, the answers to some questions may trigger the generation of additional questionnaires or amendments, that will be scheduled as new tasks.

Business processes also have associated rules and constraints to define their non-functional requirements (NFR):

1) An acknowledgement is due in less than 30 seconds after registering a task or demand for assessment.

[10]Risky situations include material facts such as not facilitating access to physically disabled people in a bank agency or having an unsecured access to the premises of the company. They can be also abstract such as the protocol used for accessing data on the company's computer server.

2) The system should be able to deal with, at least, 200 users.

3) If the number of requests exceeds 200, *FlyingPig* should implement a load balance strategy for processing the requests.

4) The privileges of the Channel-Broker must be verified *before* the execution of the actions associated to the designate user in charge π-use case.

5) The privileges of users must be verified *before* the execution of the actions associated to the answer questionnaire and add evidences π-use case.

6) All questionnaires need to be fully answered, in order to consider that a task is completed.

7) There is a time limit (in days) for each amendment required by the system.

5.2. Platform-Independent Models (PIM)

The πSOD-M models of the PIM level for the *FlyingPig* scenario are presented next.

π-UseCase Model for FlyingPig.

The π-UseCase model shown in Figure 14 describes the features and constraints for the *FlyingPig* application. In this model, three actors are identified: *Company*, *User* and *Broker* represented as stick figures. In the context of *FlyingPig*, Company is the actor requesting a risk evaluation. A Broker is responsible for coordinating the evaluation process, assigning users to be in charge of tasks as well as delegating tasks. A User, in this model is an actor who answers questionnaires according to the current situation of the Company. The User also produces evidence to support facts and performs the necessary amendments to improve the results of the risk assessment.

Each actor is associated to one or more π-use cases (depicted as white ovals in Figure 14). that describe the main functionality of the system. The π-UseCase model for *FlyingPig* defines six π-use cases.

In our model, each π-use case may be associated to one or more (non-functional) constraints (depicted as colored ovals in Figure 14). The model defines three types of constraints: *value*, *business* or *exceptional behavior*. Each constraint is identified by the word <<constraint>> followed by its type.

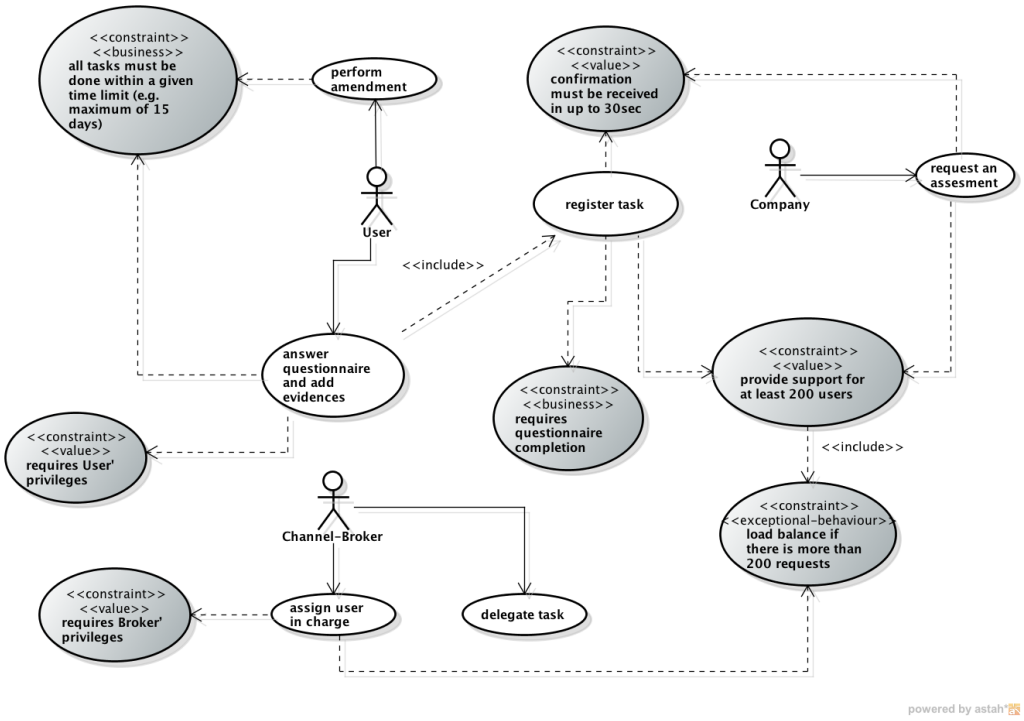

Figure 14: π-UseCase model for *FlyingPig*.

π-ServiceProcess Model for FlyingPig.

The π-ServiceProcess model (Figure 15) presents the workflow for *FlyingPig*. The actions in this model were obtained by applying the π-use case transformation rules.

The Company, Broker-Channel and User actors are transformed into lanes that represent the business collaborators. Use cases are transformed into *actions* and are represented by white boxes. The restrictions associated to each π-use case are transformed into *assertions* (represented by colored boxes) and may be decorated with pre- and post-conditions. We can see that this model refines the concepts defined in the π-UseCase model. The assertions specify those non-functional requirements, as they are seen by the actors. The next step in the development is to add these assertions to the models that specify the *FlyingPig* system.

π-ServiceComposition Model for FlyingPig.

The model presented in Figure 16 is now completed with the services that provides functions of the application *FlyingPig*The assertions in Figure 15 are implemented as *policies*. These policies express the pre- and post-conditions of the previous model. They are associated to the actions of the system.

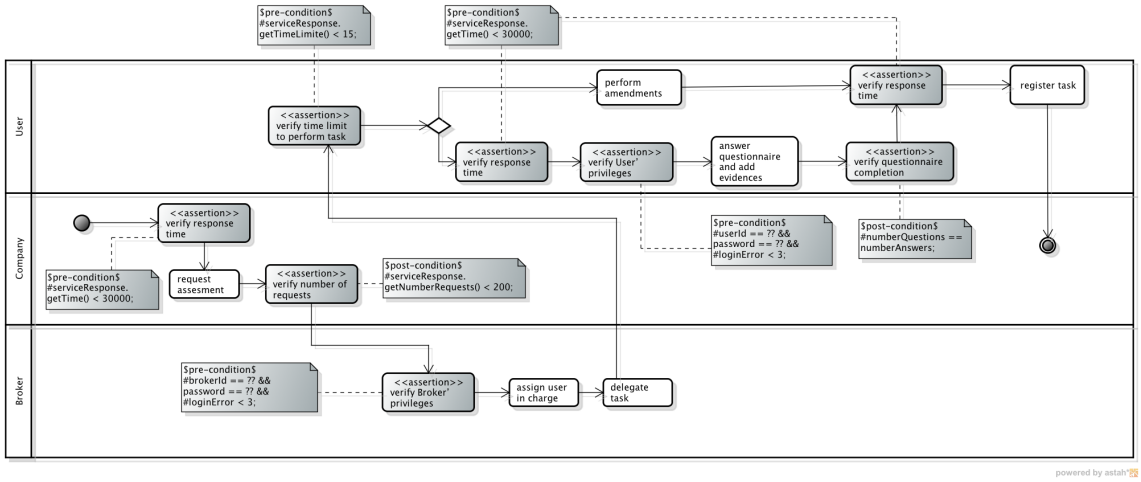

Figure 15: π-ServiceProcess model for *FlyingPig*.

π-PEWS Model for FlyingPig.

The PSM model of our case study is obtained by transforming the π-service composition model into a workflow. Notice that the workflow for *FlyingPig* is implemented by using the BPEL constructors [32], adding the notion of *policy*.

Listing 1: pi-PEWS Specification: FlyingPig.

```
 //Namespaces specify service URI
namespace orca = www.orca.mx/service.wsdl

//Operations
alias createCompanySpace = portType/createCompanySpace in orca
alias generateQuestionnaire = portType/generateQuestionnaire in orca
alias analizeAnswer = portType/analizeAnswer in orca
alias storeData = portType/storeData in orca
alias createReport = portType/createReport in orca
alias callForAmendments = portType/callForAmendments in orca

//Services
service receiveRequest(R, Id) = createCompanySpace(R, Id)
service generateNewInterface(Id, NULL) = ...
service createQuestionnaire(Id, Q) = generateQuestionnaire(Id, Q)
service notifyResponsable((Id, Q); NULL) = ...
service receiveAnswers((Id, T); P) =
        analizeAnswer((Id, T), NULL) . storeData((Id, T), NULL)
        . ((createReport(Id, P) . return(P)) + (callForAmendments(Id,T) . return(NULL)))

//Workflow
```

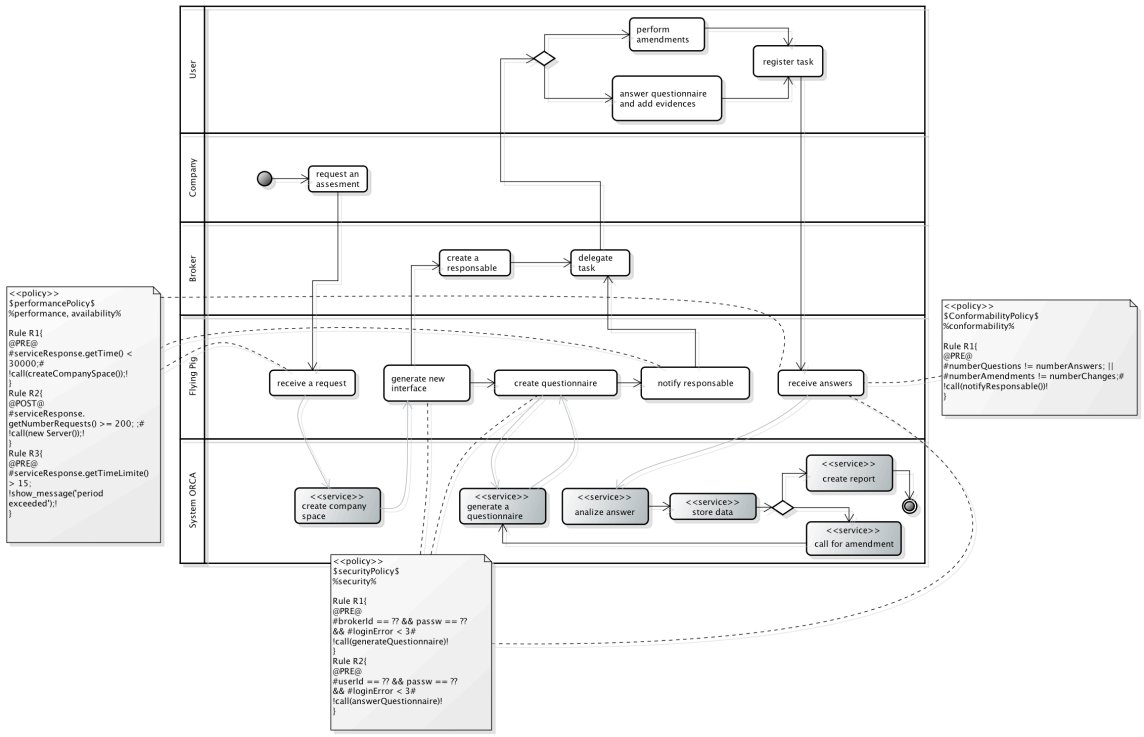

Figure 16: π-ServiceComposition model for *FlyingPig*.

receiveRequest(R, Id) 22
|| 23
(generateNewInterface(Id, NULL).createQuestionnaire(Id, Q).notifyResponsable((Id, Q); NULL)) 24
|| 25
(receiveAnswers((Id, T); P) . [P = NULL] STOP)* 26

5.3. Lessons Learned

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 service-based applications can be a simple task that implies expressing an application logic as a service composition. The challenge being ensuring the compliance between the specification and the resulting application. Software engineering methods (e.g., [8, 16, 35]) 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.

As WS-* and similar approaches, our work enables the specification and programming 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 properties 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 the 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.

6. Conclusions and future work

This paper presented πSOD-M, an MDA methodology for designing and developing reliable service-based applications. πSOD-M extends a previously defined method (called SOD-M) to include Non-Functional Requirements. These requirements are taken into account from the early stages of the software development process. Non-functional constraints are related to business rules associated to the behavior of the application and, in the case of service-based applications, they are also concerned with constraints imposed by the services.

Our methodology includes two CIM-level models, three PIM-level models and one PSM-level model. We implemented the meta-models on the Eclipse platform and we validated the approach by using an industrially inspired use case.

Our case study demonstrates the applicability of πSOD-M. The case study was developed together with our industrial partner, GCP Global. The Company is using πSOD-M for the development of their product. The case study presented here is a simplified version of their application.

References

[1] 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.

[2] Sten Andler. Predicate path expressions. In *Sixth Annual ACM Symposium on Principles of Programming Languages (6th POPL'79)*, pages 226–236, 1979.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeranwarana. Bussiness process execution language for web services. Available at http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, 2003.

[4] C. Ba, M. A. Carrero, M. Halfeld-Ferrari, and M. A. Musicante. PEWS: A New Language for Building Web Service Interfaces. *J. UCS*, 11(7):1215–1233, 2005.

[5] C. Ba, M. Halfeld-Ferrari, and M. A. Musicante. Composing web services with PEWS: A trace-theoretical approach. In *IEEE European Conference on Web Services (ECOWS)*, pages 65–74, 2006.

[6] 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.

[7] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From uml models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, January 2006.

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

[9] S. Ceri, F. Daniel, M. Matera, and F. M. Facca. Model-driven development of context-aware web applications. *ACM Trans. Internet Technol.*, 7(1), February 2007.

[10] 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.

[11] L. Chung. Representation and utilization of non-functional requirements for information system design. In *International Conference on Advanced Information Systems Engineering*, CAiSE, pages 5–30, 1991.

[12] L. Chung and J. C. Leite. Conceptual modeling: Foundations and applications. chapter On Non-Functional Requirements in Software Engineering, pages 363–379. Springer-Verlag, Berlin, Heidelberg, 2009.

[13] L. Chung, B. Nixon, E. S. K. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Springer, 1999.

[14] A. D'Ambrogio. A model-driven wsdl extension for describing the qos of web services. In *ICWS*, pages 789–796, 2006.

[15] M. V. de Castro. *Aproximacíon MDA para el Desarrollo Orientado a Servicios de Sistemas de Informacíon Web: Del Modelo de Negocio al Modelo de Composicíon de Servicios Web*. PhD thesis, Universidad Rey Juan Carlos - Escuela Técnica Superior de Ingeniería de Telecomunicación, 2007.

[16] 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.

[17] G. Di Modica, O. Tomarchio, and L. Vita. Dynamic slas management in service oriented environments. *J. Syst. Softw.*, 82(5):759–771, May 2009.

[18] V. Diamadopoulou, C. Makris, Y. Panagis, and E. Sakkopoulos. Techniques to support web service selection and consumption with qos characteristics. *J. Netw. Comput. Appl.*, 31(2):108–130, April 2008.

[19] 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.

[20] Javier Alfonso Espinosa-oviedo, Genoveva Vargas-Solar, Jos-Luis Zechinelli-Martini, and Christine Collet. Non-Functional Properties and Services Coordination Using Contracts. In *In proceedings of the 13th Int. Database Engineering and Applications Symposium (IDEAS 09)*, Cetraro, Italy, 2009. ACM.

[21] J. Fabra, V. De Castro, P. Alvarez, and E. Marcos. Automatic execution of business process models: Exploiting the benefits of model-driven engineering approaches. *Journal of Systems and Software*, 85(3):607 – 625, 2012. ¡ce:title¿Novel approaches in the design and implementation of systems/software architecture¡/ce:title¿.

[22] J. Gordijn and H. Akkermans. Value based requirements engineering: Exploring innovative e-commerce ideas. *REQUIREMENTS ENGINEERING JOURNAL*, 8:114–134, 2002.

[23] 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.

[24] 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.

[25] R. Karunamurthy, F. Khendek, and R. H. Glitho. A novel architecture for web service composition. *Journal of Network and Computer Applications*, 35(2):787 – 802, 2012.

[26] L. Li, M. Rong, and G. Zhang. A web service composition selection approach based on multi-dimension qos. In *Computer Science Education (ICCSE), 2013 8th International Conference on*, pages 1463–1468, 2013.

[27] M. Liu, M. Wang, W. Shen, N. Luo, and J. Yan. A quality of service (qos)-aware execution plan selection approach for a service composition process. *Future Generation Computer Systems*, 28(7):1080 – 1089, 2012.

[28] B. Meyer. *Object-Oriented Software Construction, 2nd Edition*. Prentice-Hall, 1997.

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

[30] R. Mohanty, V. Ravi, and M. R. Patra. Web-services classification using intelligent techniques. *Expert Syst. Appl.*, 37(7):5484–5490, July 2010.

[31] OMG. Business process model and notation (bpmn). Technical report, OMG, http://www.omg.org/spec/BPMN/2.0, 2011.

[32] OMG. Business process model and notation (bpmn). Technical report, OMG, http://www.omg.org/spec/BPMN/2.0, 2011.

[33] E. Ovaska, A. Evesti, K. Henttonen, M. Palviainen, and P. Aho. Knowledge based quality-driven architecture design and evaluation. *Information & Software Technology*, 52(6):577–601, 2010.

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

[35] 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.

[36] J. L. Pastrana, E. Pimentel, and M. Katrib. Qos-enabled and self-adaptive connectors for web services composition and coordination. *Computer Languages, Systems & Structures*, 37(1):2 – 23, 2011.

[37] A. Rumpel and K. Meissner. Requirements-driven quality modeling and evaluation in web mashups. In *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*, pages 319–322, 2012.

[38] B. Schmeling, A. Charfi, and M. Mezini. Composing non-functional concerns in composite web services. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 331 –338, july 2011.

[39] B. Selic. The pragmatics of model-driven development. *Software, IEEE*, 20(5):19–25, 2003.

[40] P. A. Souza Neto. *A Methodology for Building Service-Oriented Applications in the Presence of Non-Functional Properties*. PhD thesis, Federal University of Rio Grande do Norte - Brazil, 2012.

[41] P. A. Souza Neto, H. B. Medeiros, and R. Hallais Neto. Plugin extrator para verificação de composições pews. *HOLOS*, V.3:84–106, 2012. Avaiable in ¡¡http://www2.ifrn.edu.br/ojs/index.php/HOLOS/¿¿.

[42] 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.

[43] D. Thißen and P. Wesnarat. Considering qos aspects in web service composition. In *Computers and Communications, 2006. ISCC '06. Proceedings. 11th IEEE Symposium on*, pages 371 – 377, june 2006.

[44] H. Tran, U. Zdun, T. Holmes, E. Oberortner, E. Mulo, and S. Dustdar. Compliance in service-oriented architectures: A model-driven and view-based approach. *Information and Software Technology*, 54(6):531 – 552, 2012.

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

[46] 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.

[47] G. Yeom, T. Yun, and D. Min. Qos model and testing mechanism for quality-driven web services selection. In *Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*, pages 199–204, Washington, DC, USA, 2006. IEEE Computer Society.

[48] X. Zhang, F. Parisi-Presicce, R. S. Sandhu, and J. Park. Formal model and policy specification of usage control. *ACM Trans. Inf. Syst. Secur.*, 8(4):351–387, 2005.