πSOD-M: Building SOC Applications in the Presence of Non-Functional Requirements

Abstract: Specifying non-functional requirements (NFRs) is a complex task, being usually addressed during the latter phases of the software development process. The late inclusion of NFRs during the software development may compromise the quality of the resulting application. This paper presents πSOD-M[1], a method and associated tools that *(i)* allow the early specification of NFRs allowing users to them in an abstract way without having to care about low level details; *(ii)* embraces the MDA philosophy, generating models (code) whenever possible for reducing the specification and programming effort of the business application logic and its associated NFRs. Our solution has been used in the context of an industrial and real case study.

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

1 Introduction

In Service-Oriented Computing [35], pre-existing services are combined to build an application business logic. The selection of services is usually guided by the *functional*[a] requirements of the application being developed [9, 17, 36]. An important challenge of service-oriented development is to ensure the alignment between the functional requirements imposed by the business logic and the functions actually being developed.

Functional properties are not the only aspect to be considered 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. Ideally, Non-Functional Requirements (NFR) 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 the context of service oriented computing, the late consideration of non-functional requirements in the development process does not fully preserve the compliance and reuse expectations promoted by the service oriented paradigm.

The literature emphasizes the need for methodologies and techniques for service oriented analysis and design [35]. Existing approaches note that the convergence of model-driven software development, service orientation and business processes improvement are key for developing quality software [43]. Model Driven Development (MDD) for software systems is mainly characterized by the use of models as a product [40]. These models are

[a]Functional properties of a computer system are characterized by the effects they produce in the behavior of the system given a specific input.

successively refined from abstract specifications into actual computer programs. A recent literature review concludes that current MDD approaches rarely deal with NFRs [3][b].

Our work introduces π-*Service Oriented Development Method* (πSOD-M) to support the specification of non-functional aspects of service-oriented applications, taking into account both functional and non-functional requirements, early in the software development process[c]. Our proposal follows the MDD guidelines and proposes models, practices and techniques for the development of service-based applications. πSOD-M proposes the use of *models* to specify a software system at different levels of abstraction. Models are organized according to the guidelines of the Model Driven Architecture (MDA) [25]. The goals of πSOD-M are:

- i To improve the development process by providing an abstract view of the application and helping to ensure the conformance to its specification.
- ii To reduce the programming effort through the semi-automatic generation of models for the application, to produce concrete implementations from higher-level models.
- iii To provide a general way of describing non-functional requirements.

The applicability of our proposal was tested with an industrial case study concerning risk assessment for financial companies as implemented by the ORCA System[d]. In this work we show the application of our method πSOD-M to develop a service based application called *FlyingPig* that provides risk assessment as a service. The models presented here were generated as a result of interacting with software developers at GCP Global and trying to fulfil all the non-functional requirements present in the application.

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 and transformation rules of πSOD-M. Section 5 describes the *FlyingPig* case study that we develop for validating our method and discusses lessons learned. Finally, Section 6 concludes the paper and gives research perspectives.

2 Related Work

While Functional Requirements establish *what* is computed by an application, Non-Functional Requirements (NFRs) are concerned with *how* the task is performed. 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 [12, 14, 13]. Most research on NFRs focus on the evaluation of compliance by the software system as a whole. In service-based applications, NFRs are related to the application itself as well as to its component services.

In [7, 45] non-functional properties of web services are classified according to three perspectives: *service*, *system* and *business levels*. In [44] authors use the terms *non-functional attributes*, *composition model entity* and *model entity* to classify different

[b]The study shows that only 48 of the 129 papers considered in their systematic mapping deal with NFR. Moreover, most of them consider just specific aspects of NFR, while only 5 papers (out of 129) propose a more general approach.

[c]The letter π stands of the pronunciation of the first letter of the word "policy" which is a central concept in the representation of Non-Functional Requirementsd.

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

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 [15] 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. [39] present an approach and a toolkit for specifying and implementing web service compositions with support to several NFRs. Their approach defines abstraction levels, where the terms *Non-Functional Concerns*, *Non-Functional Attributes* and *Non-Functional Actions* are used at each level (in decreasing order of abstraction). A non-functional concern is a general term used to describe NFRs, such as *security*, *reliability* or *transactional behavior*. Each concern is refined into a set of non-functional attributes, where a non-functional attribute represents some behavior, to be refined into non-functional actions. For instance, *encryption* is a non-functional action which provides the implementation of the non-functional attribute *confidentiality*, which is part of the *security* non-functional concern.

Pastrana et al. [37] use a traditional design-by-contract approach [30]. The authors use the term *contract* to describe non-functional requirements. Contracts may present pre-conditions, post-conditions and invariants. Also, a contract may define *assertions* associated with *quality properties*. Each service may have as many associated *contracts* as needed.

Chollet et al. [11] associate (non-functional) *quality properties* to (functional) activities. They present a security meta-model that takes into account web service compositions. In this work the non-functional requirements considered are *authentication*, *integrity* and *confidentiality*. Each NFR is associated with a service activity.

The authors in [41, 46] use formal methods to define a service-based development process that takes NFRs into account. In [1, 37] ontologies are used to define and model NFRs, as in [44, 24] Business Process Modeling is used for system specification, including NFRs. In the method defined in [44], each task and data item of the application can be annotated with functional as well as 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 [41] presents steps to select services by taking QoS information into account: *(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 authors consider *performance*, *cost*, *reliability* and *availability*.

Karunamurthy et al. [27] define NFRs, such as *cost*, *response time*, *availability*, *security*, *reliability* and *reputation*, as *non-function parameters*. The *Non-Functional Specification Language* is proposed as a domain specific language to express *non-function parameters*.

Liu et al. [29] use the term *QoS parameter* to describe NFRs such as *cost*, *execution duration*, *accuracy*, *security*, *integrity*, *availability* and *reliability*. In the same way, Tran et al. [42] use the term *QoS policies* to classify similar non-functional requirements.

Li et al. [28] 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. [38] associate *quality requirements* to *quality properties*. Quality requirements are to be specified as constraints.

Ceri et al.[10] 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. [34] 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* [46, 8, 26], *property* [22], *factor* [31, 24], *characteristic* [19], *quality level* [18], and *value* [41, 8].

Despite of the different notations found in the literature for classifying NFRs, some NFRs are frequently considered, such as *security*, *performance*, *reliability*, *usability*, and *availability*. However, distinct hierarchies and models are proposed for NFRs, according to different perspectives. We have identified a number of approaches [15, 11, 39, 8, 22, 34] that use MDD (Model Driven Development) for designing and developing applications.

Besides the existence of these works, it is important to remark that authors continue emphasizing the limited attention of the model-driven proposals for service development to NFRs [3, 2].As Ameller et al. notes in a recently published literature review, there are *"Not many papers that presented a generic approach to deal with NFRs"*. Those authors also remark the lack of industrial use cases [3].

To the best of our knowledge, and based on the related work analysis, there are no proposals that define a service-oriented approach for the whole development process of systems, considering non-functional requirements. Despite the efforts made to support the new technological proposals for the Web, such as web services, these development approaches address their development processes according to traditional software engineering, with emphasis on the functional aspect of the application.

Our method for developing service-oriented applications aims to overcome limitations of existing proposals by dealing with NFRs. Our approach targets the characteristics of 3 areas[e] to propose a method for developing service-oriented applications considering non-functional aspects. The motivation of our method is to target the specification of non-functional aspects in the early stages of development of service-oriented applications. Our approach specifies *policies* for modeling the non-functional requirements of a service. It separates the specification of non-functional requirements from the main functionalities of the application. This is interesting because once the policies for a given application have been defined they can be reused and/or specialized for another application with similar requirements. In order to show these characteristics this paper introduces a validation use case, which correponds to a service based application provided by the company ORCA. The application is related to the risk assessment industry.

3 Modeling Reliable Service Compositions with πSOD-M

In this section we present πSOD-M, an MDD-based method for building service compositions with NFRs. Our work 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*

[e](i) Non-Functional Requirements; (ii) Methodology for Service-Oriented Development; and (iii) Model-Driven Engineering

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 purpose of this section is to present the πSOD-M meta-models, from higher to lower level of abstraction, along with a running example. The transformations between models are presented next, in section 4.

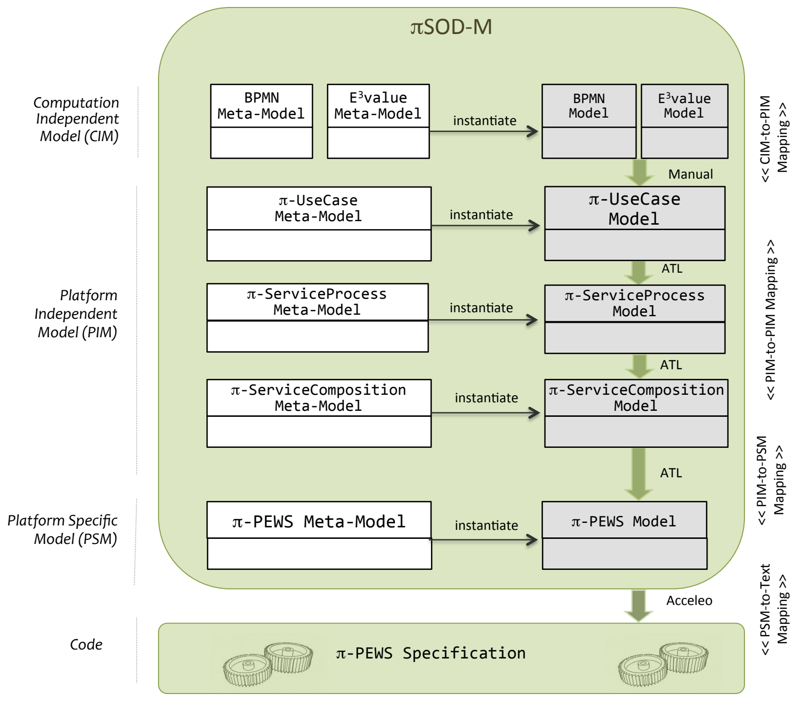


Figure 1 πSOD-M Overview.

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* [23] and *BPMN* [32] meta-models for this purpose.

3.1.1 E^{3}value Meta-Model

The e^{3}value model identifies the value/information exchange between system components (see Figure 2). This model represents a business case graphically as a set of value exchanges (∇ $\triangle$) and value activities (rounded boxes)[f] performed by business actors (squared boxes). The model is well suited to enhance the understanding of the environment in which an

[f]According to [23], a value activity is defined as an activity expected to be profitable to one or more actors.

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. Dependency paths may also contain *OR* and *AND* elements (both for initiate and join alternative and parallel paths).

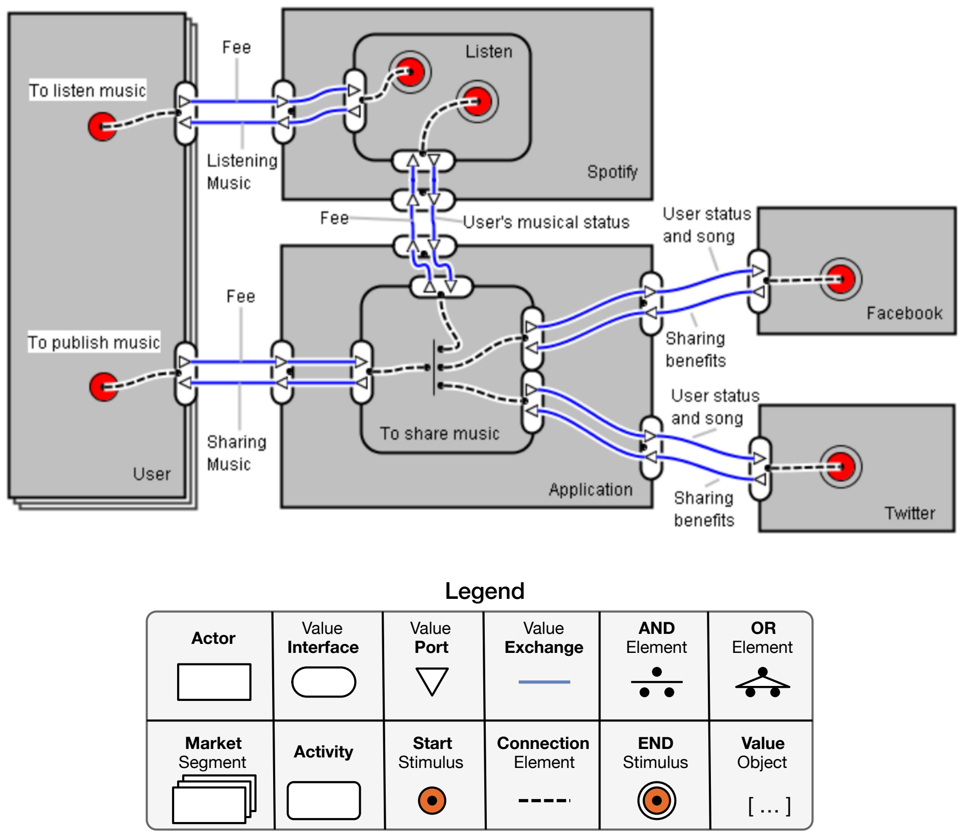


Figure 2 e3value model for "To Publish Music".

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 service Spotify exports a method for obtaining information about the music a given user is listening:* get-Last-Song (userid): String*;*
- *The services Facebook and Twitter 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 model shows Spotify and a private application (which is also a service) that provide services for listening and publishing information about music being listened by users. The private application interacts with Spotify for obtaining information about the flow of music being listened by a user in return of a fee for a 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. However, the e^{3}value model does not help to understand the business process of the application nor the conditions in which the different steps of this process are executed. In order to face this shortcoming, our method proposes the use of the BPMN meta-model as a tool for modeling this aspect of the application.

3.1.2 BPMN Meta-Model

BPMN [32] is a graphical representation that establishes the business process of the application through a high-level workflow[g]. The next example illustrates the use of this meta-model.

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

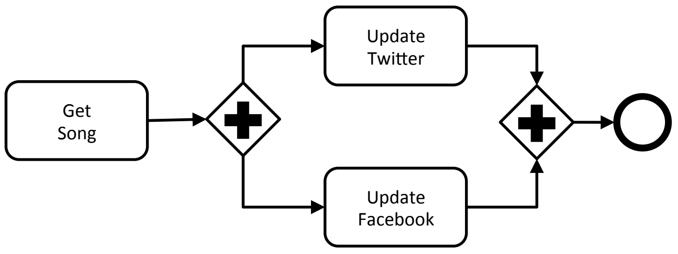


Figure 3 BPMN model for "To Publish Music".

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 πSOD-M, as described next. Notice that 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, abstracting the details of any particular platform. The specification defines those parts of the system that do not change from one

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

platform to another. Our method defines three PIM-level meta-models: π-*UseCase*, π-*ServiceProcess* and π-*ServiceComposition*.

3.2.1 π-UseCase Meta-Model

Figure 4 shows the π-*UseCase* meta-model. This model is used to give the first representation of the application in terms of functionality, and of its non-functional requirements. The notion of *policy* is used to describe NFRs. Figure 4 highlights the concepts related to NFRs. The π-*UseCase* meta-model extends the UML Use Case meta-model to describe NFRs with the following concepts: Business service[h], 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), as 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 by Packages. A Business Collaborator denotes an external service or a system that interacts with the application that is being modelled. Each Business Collaborator combines the features described in each Package.

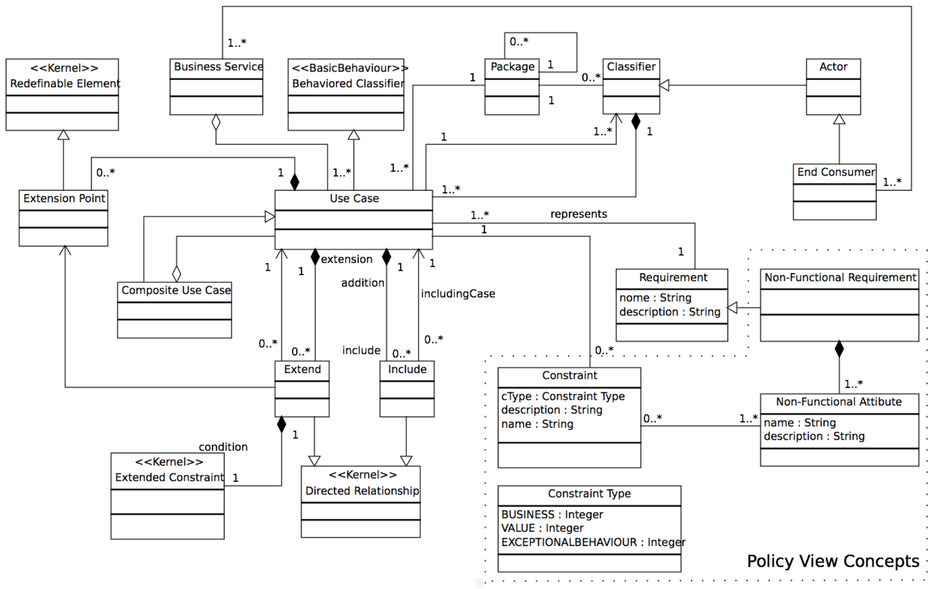


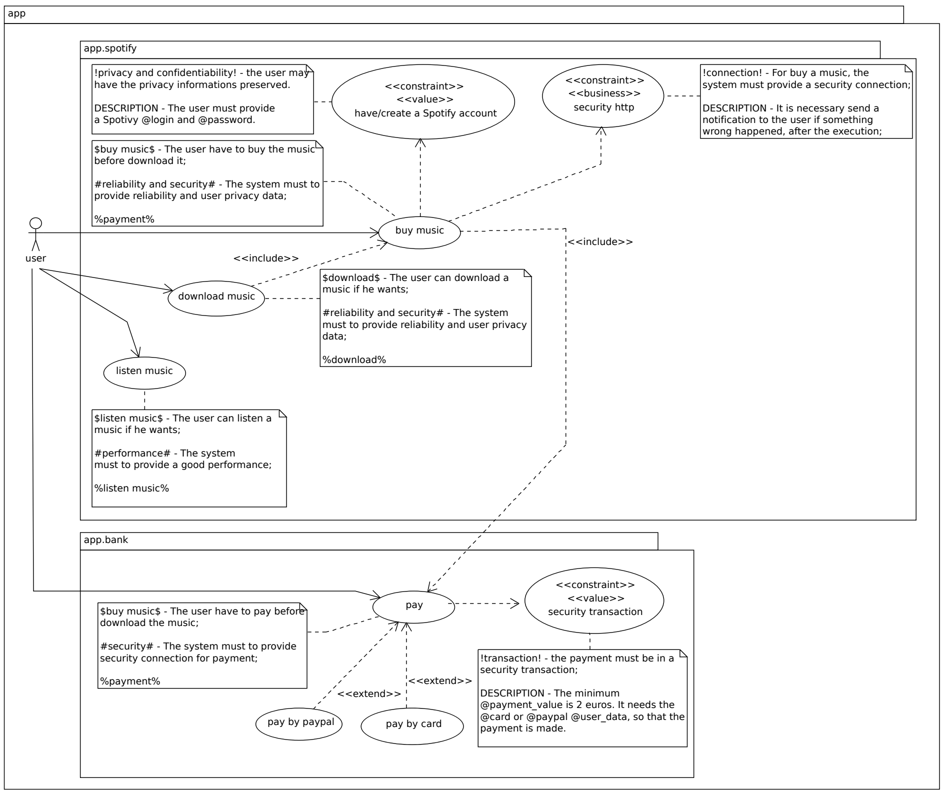
Figure 4 π-UseCase Meta-Model.

The Non-functional requirement and Non-functional Attribute concepts are represented by Use Cases and Constraints. A Use Case may have several Constraints. Each Constraint has a name, description, and a flag to indicate that it has to be dynamically checked. Each Constraint is represented as a stereotyped ⟨constraint⟩[i] use case.

[h] We use the small caps font for referring to classes of a meta-model.
[i] We use the sans serif font for referring to classes defined using a meta-model.

There are three kinds of constraints: *(i)* A data pair (Value Constraint), represented as the stereotype "value"; *(ii)* Business rules (Business Constraint), represented as the stereotype "business"; and *(iii)* An Exceptional Behaviour constraint, represented by the stereotype "exceptional_behavior". The use of these constraints is shown in the next example.

Example 3 (To Publish Music *(cont)*) *Figure 5 shows a π-UseCase model for our example. 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 service usage – for example, the fact that both Facebook and Twitter require authentication in order to call their methods for updating the wall; (ii) 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).*


app
app.spotify
<<constraint>> <<value>> have/create a Spotify account
<<constraint>> <<business>> security http
user
buy music
<<include>>
download music
<<include>>
listen music
app.bank
pay
<<constraint>> <<value>> security transaction
<<extend>>
<<extend>>
pay by paypal
pay by card

!privacy and confidentiability! - the user may have the privacy informations preserved.

DESCRIPTION - The user must provide a Spotivy @login and @password.

!connection! - For buy a music, the system must provide a security connection;

DESCRIPTION - It is necessary send a notification to the user if something wrong happened, after the execution;

$buy music$ - The user have to buy the music before download it;

#reliability and security# - The system must to provide reliability and user privacy data;

%payment%

$download$ - The user can download a music if he wants;

#reliability and security# - The system must to provide reliability and user privacy data;

%download%

$listen music$ - The user can listen a music if he wants;

#performance# - The system must to provide a good performance;

%listen music%

$buy music$ - The user have to pay before download the music;

#security# - The system must to provide security connection for payment;

%payment%

!transaction! - the payment must be in a security transaction;

DESCRIPTION - The minimum @payment_value is 2 euros. It needs the @card or @paypal @user_data, so that the payment is made.

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

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 user 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 by a value stereotype. 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 in the *π-UseCase*. In Figure 6, the concepts of the *π-ServiceProcess* meta-model are: Contract, Assertion, Exceptional behavior, Activity, Service Activity, Action and Constraint. The dotted part of the figure defines those concepts related to the representation of NFRs.

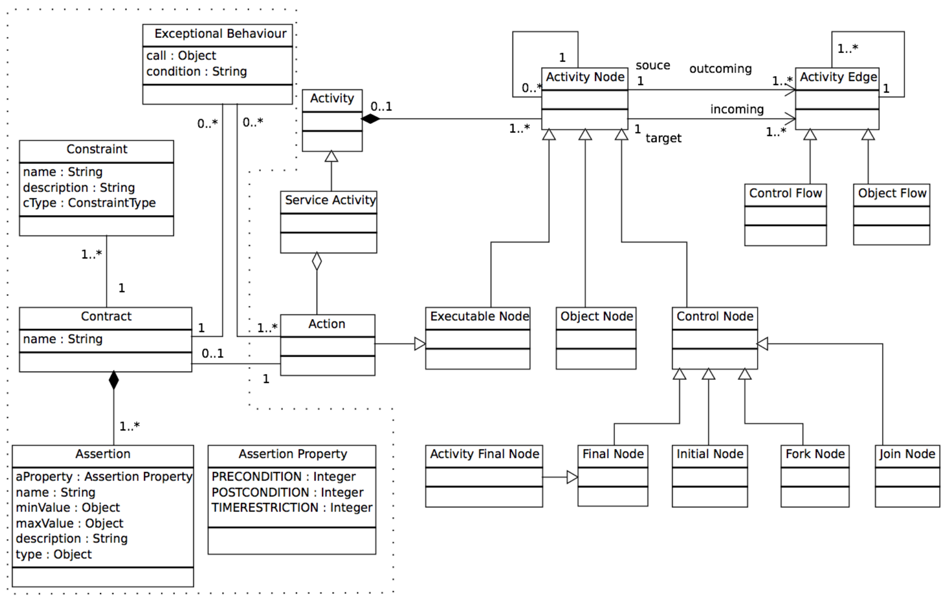


Figure 6 π-Service Process Meta-Model.

This model shows the set of logically related activities to be performed in a service-based application. The activities of this model represent a behavior that can implement 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- and post-conditions. The service that fulfills the Contract and its Assertions will be chosen and used in the execution flow. This part of the process requires the verification of some logic conditions. In the present version of the tool, this step requires human intervention.

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 about the payment authorization was sent both to the user and to Spotify. 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.*

3.2.3 π-ServiceComposition Meta-Model

In Figure 8, the π-ServiceComposition meta-model provides meta-classes to represent workflowsthat model business processes. The *π-ServiceComposition* 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 single policy. This meta-model defines:

- A Business Collaborator meta-class, to represent the classes of entities that collaborate in business processes by performing some action. An instance of this meta-class is graphically 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 of the collaborator is set to **true**.
- Actions, a kind of ExecutableNode, are represented 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, composed by executable nodes.
- In order to represent constraint types associated to service compositions, we defined 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* [20, 21]. An *A-policy* is defined by attributes and rules. The conditions of each rule are evaluated. In case of failure, the actions of the rule will be performed. The meta-class Rule represents event-condition-action rules where the Event part represents the moment in which a constraint is evaluated. An *A-policy* defines variables and operations

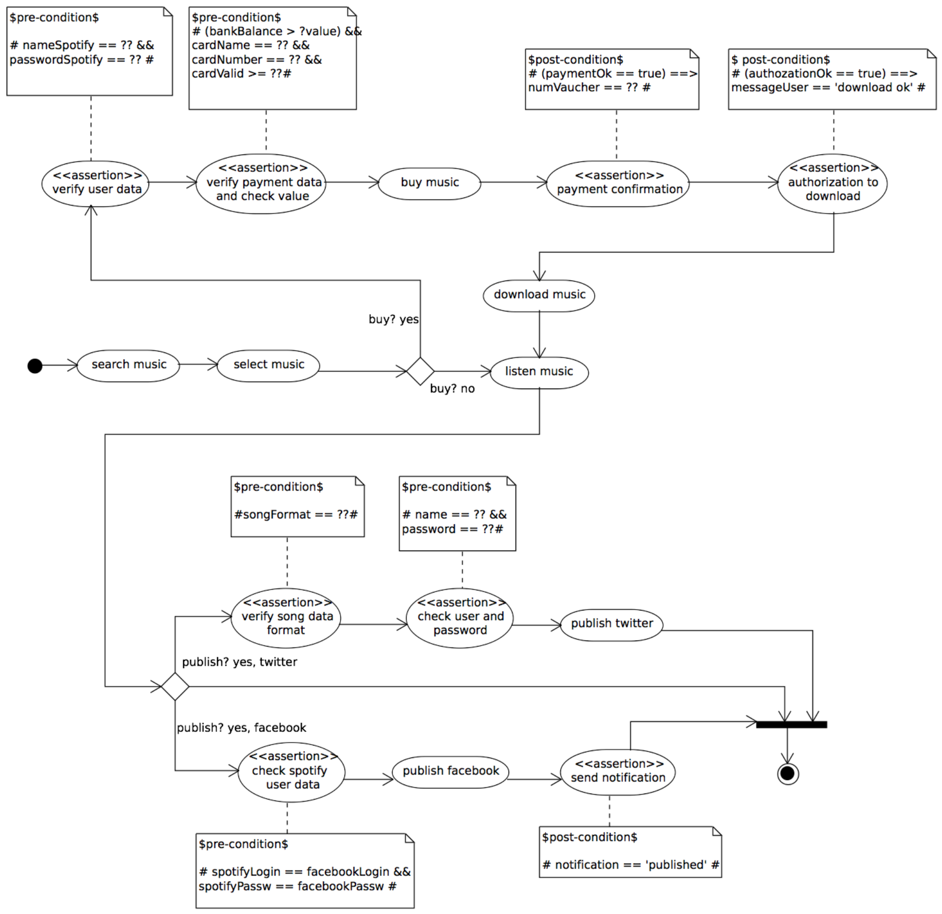


Figure 7 π-ServiceProcess model for “To Publish Music”.

that can be shared by the rules and that can be used for expressing their Event and Condition parts.

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 with three service activities:* 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 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* UpdateTwitter *(see*

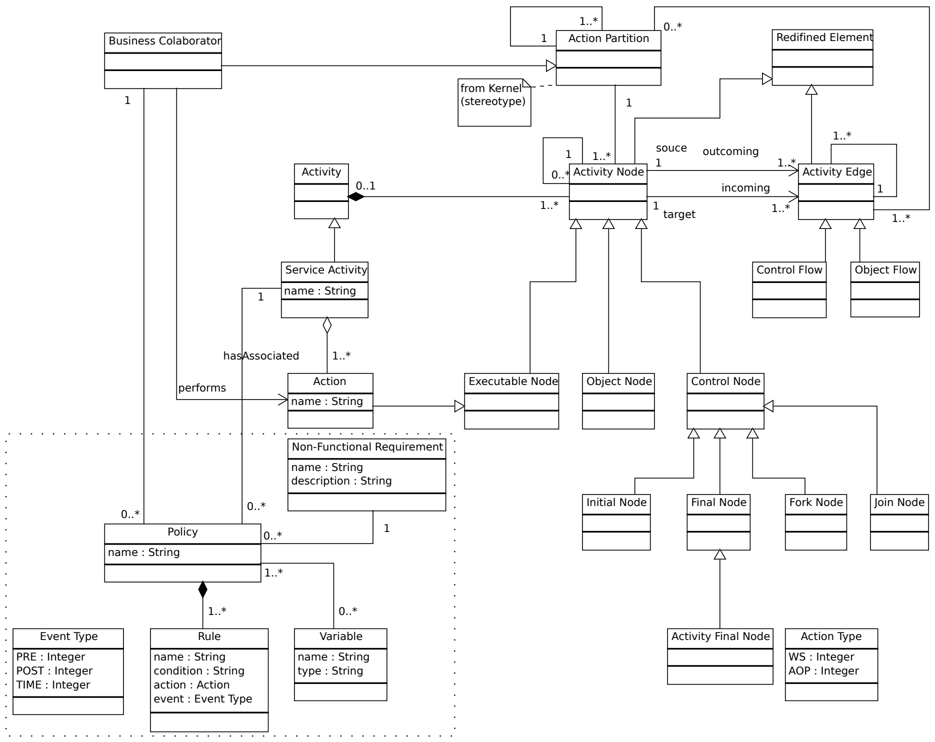


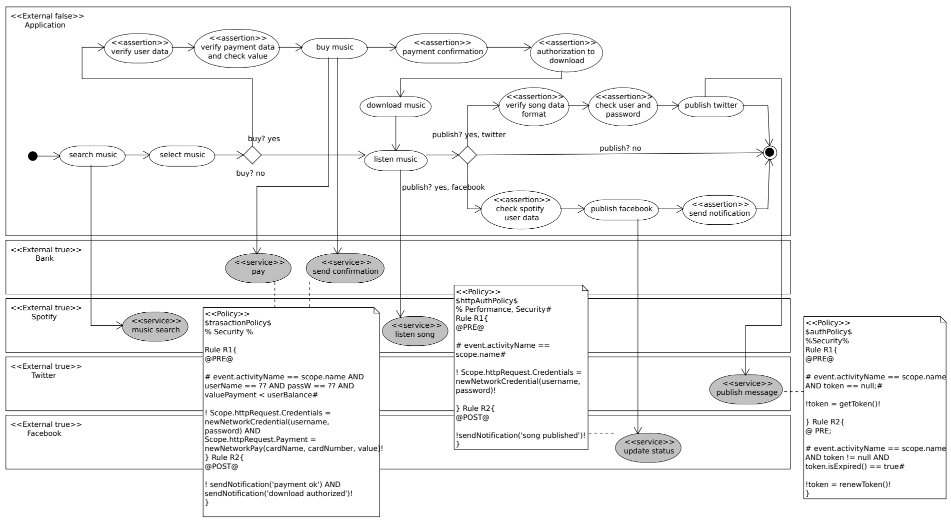
Figure 8 π-Service Composition Meta-model.

Figure 9). In the same way, the Facebook *class* HTTPAuthPolicy, *for the http authentication protocol is associated to the activity* UpdateFacebook. *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 is defined, it can be transformed into a lower level model (in our case, π-PEWS) to support code generation as 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 put together the platform-independent view with the specific



```
<<Policy>>
$trasactionPolicy$
% Security %

Rule R1{
@PRE@

# event.activityName == scope.name AND
userName == ?? AND passW == ?? AND
valuePayment < userBalance#

! Scope.httpRequest.Credentials =
newNetworkCredential(username,
password) AND
Scope.httpRequest.Payment =
newNetworkPay(cardName, cardNumber, value)!
} Rule R2{
@POST@

! sendNotification('payment ok') AND
sendNotification('download authorized')!
}
```

```
<<Policy>>
$httpAuthPolicy$
% Performance, Security#
Rule R1{
@PRE@

# event.activityName ==
scope.name#

! Scope.httpRequest.Credentials =
newNetworkCredential(username,
password)!

} Rule R2{
@POST@

!sendNotification('song published')!
}
```

```
<<Policy>>
$authPolicy$
%Security%
Rule R1{
@PRE@

# event.activityName == scope.name
AND token == null;#

!token = getToken()!

} Rule R2{
@ PRE;

# event.activityName == scope.name
AND token != null AND
token.isExpired() == true#

!token = renewToken()!
}
```

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

aspects of the platform to implement the system. These models can be automatically translated into actual computer programs. We have defined one meta-model at this level.

π-*PEWS* provides concepts for modelling 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 [5, 33].PEWS [6]is a notation to express service compositions. The language is based on the notion of Path Expressions [4] and can easily be translated into any actual composition language, such as BPEL [5]. Figure 10 presents the π-Pews meta-model, where we identify classes to describe:

- Service compositions: Namespace represents the interface exported by a service, Operation represents a call to a service method, CompositeOperation, Operator and Path denote 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 service compositions: A-Policy, Rule, Event, Condition, Action, State, and Scope.

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 example. The code contains namespaces and definitions (obtained from the business collaborators of the π-SCM model), a workflow expression (Path) containing operation calls and contracts (derived from the A-Policies).*

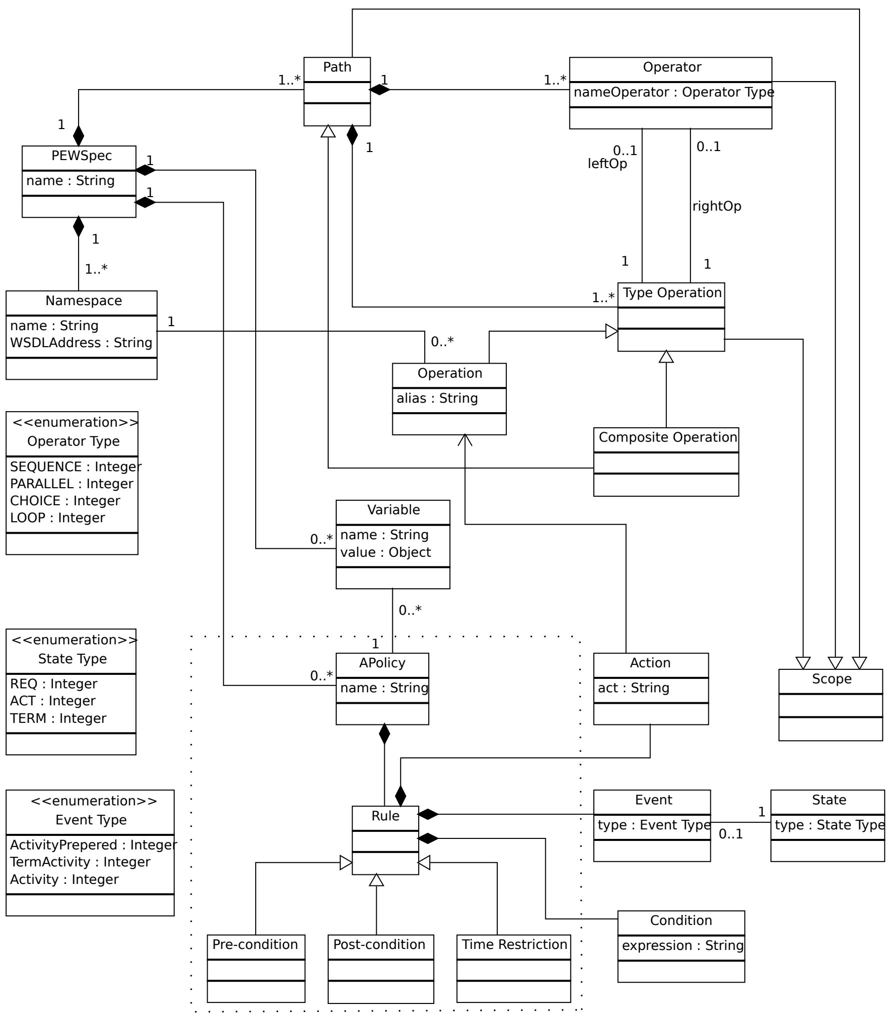
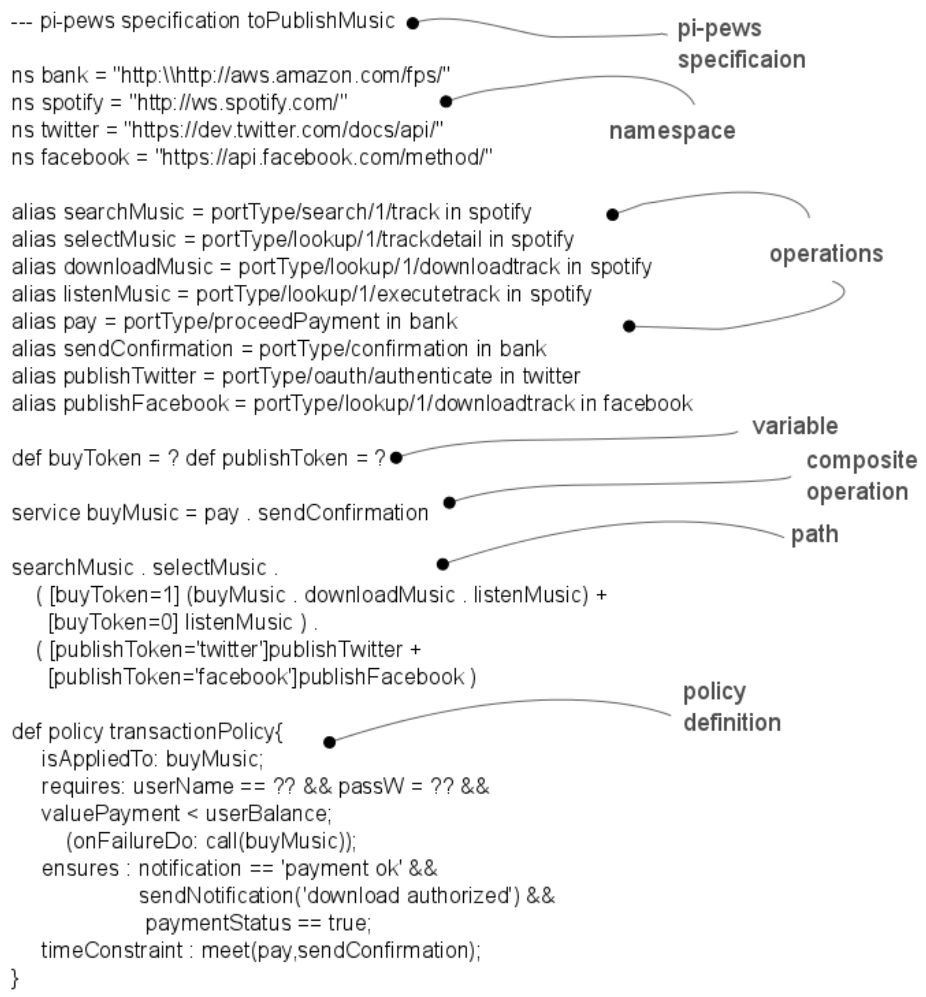


Figure 10 π-Pews Meta-model.

4 Transformation Rules

πSOD-M proposes intra-level transformation rules from π-use case to π-service process models, π-service process to π-service composition models, as well as inter-level transformation rules from π-service composition to π-PEWS models.

These rules were implemented into a semi-automatic tool, the π-SOD-M plug in for Eclipse. At each stage of transformation it is possible to include new constraints. Transformation in the more abstract stages is completely automatic (i.e., from π-UseCase to π-ServiceProcess, and from π-ServiceProcess to π-ServiceComposition). In contrast, the transformation to specific platforms (i.e. from π-ServiceComposition to π-PEWS), the analyst must intervene and check the generation of the specification code because there are programming details that can be interpreted in different manners depending on the target platform. In this sense, the method is more appropriate for those who have background



```
--- pi-pews specification toPublishMusic

ns bank = "http:\\http://aws.amazon.com/fps/"
ns spotify = "http://ws.spotify.com/"
ns twitter = "https://dev.twitter.com/docs/api/"
ns facebook = "https://api.facebook.com/method/"

alias searchMusic = portType/search/1/track in spotify
alias selectMusic = portType/lookup/1/trackdetail in spotify
alias downloadMusic = portType/lookup/1/downloadtrack in spotify
alias listenMusic = portType/lookup/1/executetrack in spotify
alias pay = portType/proceedPayment in bank
alias sendConfirmation = portType/confirmation in bank
alias publishTwitter = portType/oauth/authenticate in twitter
alias publishFacebook = portType/lookup/1/downloadtrack in facebook

def buyToken = ? def publishToken = ?

service buyMusic = pay . sendConfirmation

searchMusic . selectMusic .
   ( [buyToken=1] (buyMusic . downloadMusic . listenMusic) +
    [buyToken=0] listenMusic ) .
   ( [publishToken='twitter']publishTwitter +
    [publishToken='facebook']publishFacebook )

def policy transactionPolicy{
    isAppliedTo: buyMusic;
    requires: userName == ?? && passW = ?? &&
    valuePayment < userBalance;
       (onFailureDo: call(buyMusic));
    ensures : notification == 'payment ok' &&
                sendNotification('download authorized') &&
                 paymentStatus == true;
    timeConstraint : meet(pay,sendConfirmation);
}
```

Figure 11 π-PEWS Specific and Policy Representation.

knowledge in Information Technology.. The transformations from CIM to PIM level models are not automatized, due to the informal nature of the CIM level.

4.1 From π-UseCase to 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 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 [16].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 specifies 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.

Example 7 (To Publish Music *(cont)*) *The transformation rules have been applied to the model in Figure 5 to obtain the π-Service Process model for our example (Figure 7).*

The "listen music" use case is transformed into a Service Action that represents a Spotify function to be invoked to play 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 use case "download music" 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 transformation of a π-Service Process model into a π-Service Composition model groups 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 9 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 transforming A-Policies into Contracts.

Single actions (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 define a (named) workflow of the application. Composition patterns expressed using *merge, decision, fork* and *join* are transformed into their corresponding workflows of the π-PEWS model.

A-Policies defined for the entities of a π-service composition model generate 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 Variable, with Name and Type attributes 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 11 shows the π-PEWS code resulting from the π- service composition model (Figure 9) 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)[j] to implement the π-Service Composition and π-Pews meta-models. From these meta-models, we developed plug-ins to support their graphical representation.
- We used ATL[k] for implementing the mappings between models.
- We used Acceleo[l] to implement the code generation plug-in to generate executable code. It takes a π-PEWS model and generates the code to be executed by the *A-Policy* based service composition execution environment.

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

[k]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.

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

5 Applying πSOD-M: The *FlyingPig* Case Study

To validate the applicability of our method we have developed a case study concerning risk assessment for financial companies as implemented by the ORCA System[m].Risk assessment is implemented by an interactive business process based on the exchange of questionnaires used 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 to access 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, 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 (*FlyingPig*), for providing risk assessment as a service. In order to provide this functionality, *FlyingPig* benefits from ORCA's legacy services providing storage, assessment and data visualization functions. In the next sections we apply πSOD-M to develop the *FlyingPig* risk assessment system. The models presented 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*. 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). In the next step, ORCA suggests amendments (recommendations to change practices) and provides an evaluation report.

Figure 13 shows the BPMN model for the *FlyingPig* scenario. This model is partitioned for better understanding the value exchanging process. 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

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

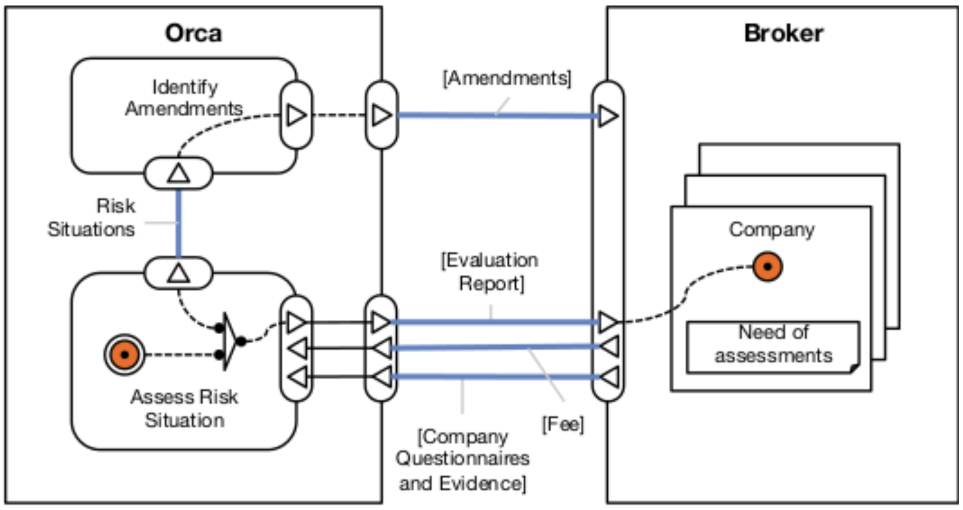


Figure 12 E^{3}value model for *FlyingPig*.

users that will answer questionnaires to evaluate risk. Other tasks include amending a “risky situation” as well as producing evidence to show that a specific risk has been eliminated[n].

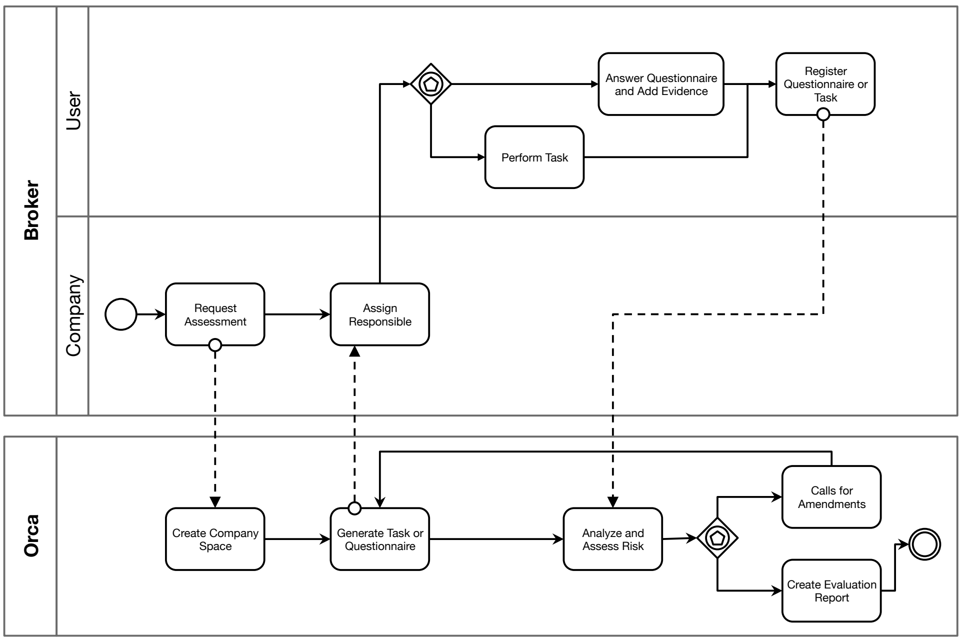


Figure 13 BPMN model for *FlyingPig*.

[n]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 inmaterial restrictions such as the protocol used for accessing data on the company’s computer server.

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 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.
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 a task as completed.
7. There is a time limit (in days) for amendments 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. These models were manually obtained from the CIM level models.

5.2.1 π-UseCase Model for FlyingPig

The π-UseCase model shown in Figure 14 describes the features and constraints of the *FlyingPig* application. In this model, three actors are identified: *Company*, *User* and *Broker* represented as stick figures. Company is the actor requesting a risk evaluation as a Broker is responsible for coordinating the evaluation process, assigning users to tasks as well as delegating tasks. A User 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 π-UseCases (white ovals in Fig. 14) that describe the main functionality of the system. The π-UseCase model for *FlyingPig* defines six π-UseCases. Each π-UseCase may be associated to (non-functional) constraints (coloured ovals in Fig. 14).Three types of constraints are defined: *value*, *business* or *exceptional behavior*. Constraints are identified by the word <<constraint>> followed by its type.

5.2.2 π-ServiceProcess Model for FlyingPig

The π-ServiceProcess model presents the workflow for *FlyingPig* (Figure 15). 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 π-use cases 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.

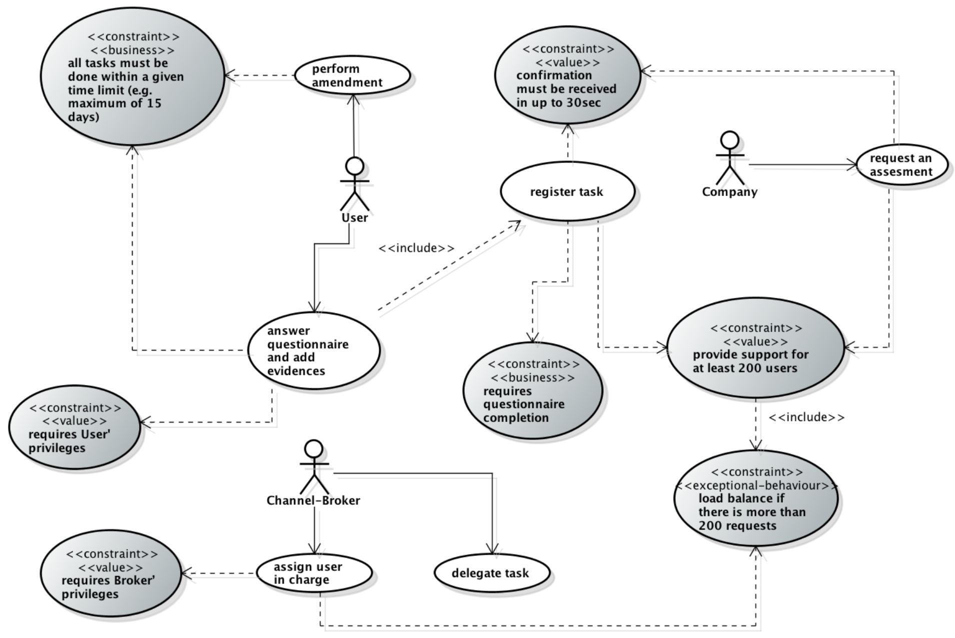
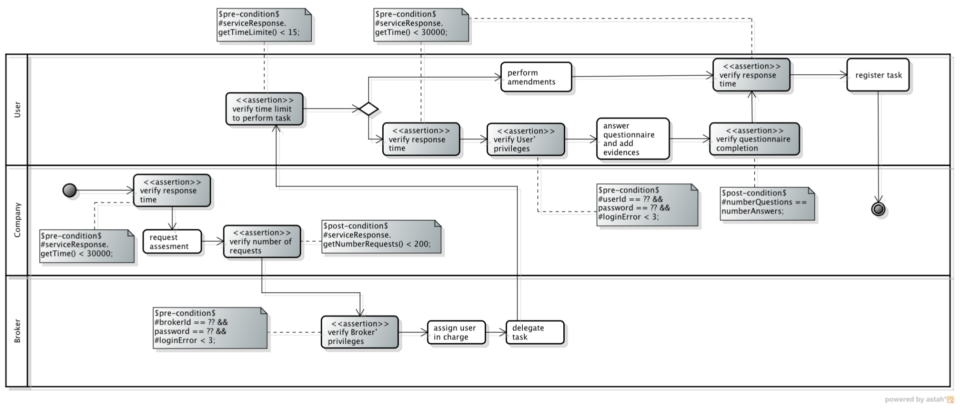

<<constraint>>
<<business>>
all tasks must be done within a given time limit (e.g. maximum of 15 days)
perform amendment
<<constraint>>
<<value>>
confirmation must be received in up to 30sec
User
register task
Company
request an assesment
<<include>>
answer questionnaire and add evidences
<<constraint>>
<<business>>
requires questionnaire completion
<<constraint>>
<<value>>
provide support for at least 200 users
<<include>>
<<constraint>>
<<value>>
requires User' privileges
Channel–Broker
<<constraint>>
<<exceptional–behaviour>>
load balance if there is more than 200 requests
<<constraint>>
<<value>>
requires Broker' privileges
assign user in charge
delegate task

Figure 14 π-UseCase model for *FlyingPig*.



```
$pre-condition$
#serviceResponse.
getTimeLimite() < 15;
```

```
$pre-condition$
#serviceResponse.
getTime() < 30000;
```

```
$pre-condition$
#userId == ?? &&
password == ?? &&
#loginError < 3;
```

```
$post-condition$
#numberQuestions ==
numberAnswers;
```

```
$pre-condition$
#serviceResponse.
getTime() < 30000;
```

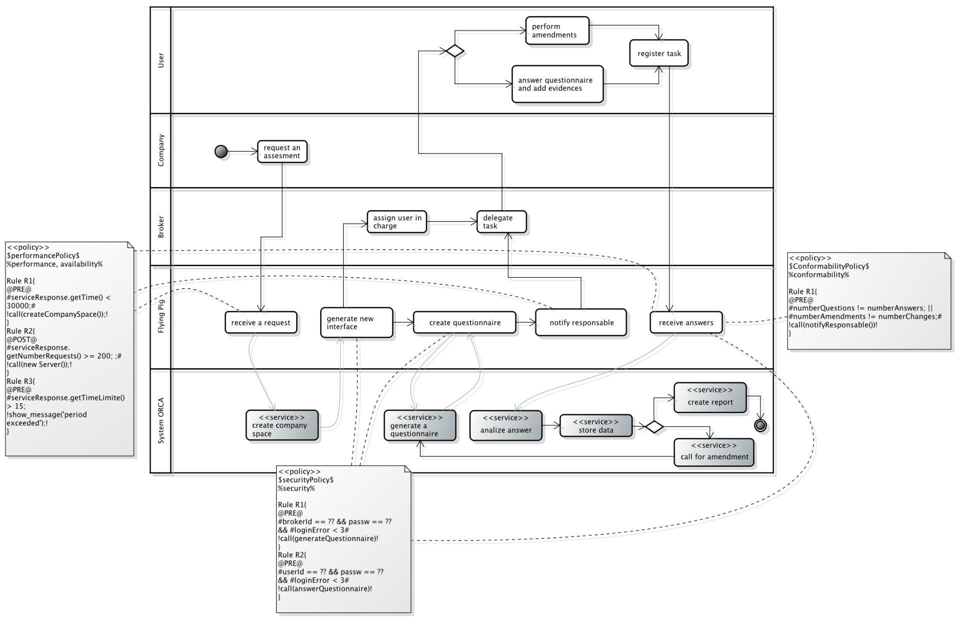
```
$post-condition$
#serviceResponse.
getNumberRequests() < 200;
```

```
$pre-condition$
#brokerId == ?? &&
password == ?? &&
#loginError < 3;
```

Figure 15 π-ServiceProcess model for *FlyingPig*.

5.2.3 *π-ServiceComposition Model for* FlyingPig

The model in Figure 16 shows the services that provide the functions of *FlyingPig*. The assertions in Figure 15 are implemented as *policies*. These policies express the pre- and post-conditions of the previous model, being associated to the actions of the system.



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

Rule R1{
@PRE@
#serviceResponse.getTime() <
30000;#
!call(createCompanySpace());!
}
Rule R2{
@POST@
#serviceResponse.
getNumberRequests() >= 200; ;#
!call(new Server());!
}
Rule R3{
@PRE@
#serviceResponse.getTimeLimite()
> 15;
!show_message('period
exceeded');!
}
```

```
<<policy>>
$ConformabilityPolicy$
%conformability%

Rule R1{
@PRE@
#numberQuestions != numberAnswers; ||
#numberAmendments != numberChanges;#
!call(notifyResponsable())!
}
```

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

Rule R1{
@PRE@
#brokerId == ?? && passw == ??
&& #loginError < 3#
!call(generateQuestionnaire)!
}
Rule R2{
@PRE@
#userId == ?? && passw == ??
&& #loginError < 3#
!call(answerQuestionnaire)!
}
```

Figure 16 π-ServiceComposition model for *FlyingPig*.

5.2.4 π-PEWS Model for FlyingPig

The PSM of our case study is given in Figure 17. This model is obtained by transforming the π-service composition model into a workflow. Note that this workflow is implemented by using BPEL constructors [33] and *policy* notions.

```
//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
receiveRequest(R, Id)
|| (generateNewInterface(Id, NULL)
 . createQuestionnaire(Id, Q) . notifyResponsable((Id, Q); NULL))
|| (receiveAnswers((Id, T); P) . [P != NULL] STOP)*
```

Figure 17 π-PEWS Model for *FlyingPig*.

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 such as security, isolation, adaptability, atomicity, and more. These non-functional properties must be ensured at execution time.

In the context of service-oriented applications, ensuring non-functional properties is challenging due to the nature of their components. Those components are defined by APIs which do not necessarily export information about their internals.

Given a set of services with their exported methods, building service-based applications may consist on expressing an application logic as a service composition. During this task, we must ensure the compliance between the specification and the resulting application. Software engineering methods (e.g., [9, 17, 36]) 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, we specify policies for a service composition in an orthogonal way. Besides, these approaches suppose that non-functional properties are implemented according to 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, the policies defined for a given application can be reused and/or specialized for another one with the same requirements or that uses services that impose the same constraints.

Wrapping up, the industrial use case presented here provided a real validation context for testing the usability of our method πSOD-M. The software units acting as services provide specific functions and constraints that were combined to build the business logic of the FlyingPig application. This application is representative of the kind of target systems of πSOD-M. The different types of business rules could be modelled as first class citizens and clearly associated to the business logic of the system. Thereby, we could show how non-functional properties can be associated both to the components (i.e., services) and to the global business logic of the system. πSOD-M allowed to derive these non-functional properties thanks to the specialization and transformation of the different meta-models.

We learned and showed that decoupling the description (specification) of non functional properties from the functional aspects, allows developers to focus on the logic of their application, isolating them from the non-functional aspects. Furthermore, it improves modularity by allowing the same application to be run under different non-functional settings, if needed. This can be interesting if business rules change or if constraints associated to services evolve.

Finally, the system FlyingPig had been incrementally built using classic software engineering techniques before applying πSOD-M. It is difficult to compare the initial time to market versus the time to market using our method. The company could not estimate a realistic value of this measure for the first version of the application. We intend to evaluate the quality of experience of programmers, when they will have to build a new version of FlyingPig. The quality of experience related to πSOD-M will be given by measuring (i) time to market when a new module will be designed and implemented in the system (ii) maintainability of the new module having a separated and clear vision of business logic and non-functional properties. Both two measures are important to the company.

6 Conclusions and Future Work

We presented πSOD-M, a model-driven method 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 method 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 was developed together with our industrial partner GCP Global to demonstrate the applicability of πSOD-M. The Company is using πSOD-M for the development of their product. The case study presented is a simplified version of their application.

In the future, we expect to: *(i)* select metrics for assessing the efficacy of πSOD-M; *(ii)* use the feedback provided by our industrial partner to improve the methodology; *(iii)* assess the use of πSOD-M in other contexts, in order to investigate the generality of the approach.

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] D. Ameller, X. Franch, and J. Cabot. Dealing with non-functional requirements in model-driven development. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 189–198, Sept 2010.

[3] David Ameller, Xavier Burgués, Oriol Collell, Dolors Costal, Xavier Franch, and Mike P. Papazoglou. Development of service-oriented architectures using model-driven development: A mapping study. *Information and Software Technology*, 62(0):42 – 66, 2015.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] A. D'Ambrogio. A Model-driven WSDL Extension for Describing the QoS of Web Services. In *ICWS*, pages 789–796, 2006.

[16] 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 (in Spanish).

[17] 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.

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

[19] 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.

[20] 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.

[21] Javier Alfonso Espinosa-oviedo, Genoveva Vargas-Solar, José-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.

[22] 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. Novel approaches in the design and implementation of systems/software architecture.

[23] J. Gordijn and H. Akkermans. Value based requirements engineering: Exploring innovative e-commerce ideas. *Requirements Engineering Journal*, 8:114–134, 2002.

[24] 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 51.12, pages 1712–1738, Newton, MA, USA, December 2009. Butterworth-Heinemann.

[25] J. Miller and J. Mukerji. MDA Guide Version 1.0.1, 2003.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

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

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

[33] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, April 2007.

[34] 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.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

[41] 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.

[42] 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.

[43] A. Watson. A brief history of MDA. *CEPIS UPGRADE: The European Journal for the Informatics Professional*, IX(2):7–11, April 2008.

[44] 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.

[45] 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.

[46] 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.