# Supporting Non-Functional Requirements for Services Software Process: An MDD Approach

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

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

**Abstract.** This paper presents an extension to the Service Oriented Development Method (SOD-M) to support the representation of some non-functional requirements. Specifically, we propose to extend SOD-M with: *(i)* meta-models for representing non-functional requirements in different abstraction levels; *(ii)* model to model transformation rules, useful to semi-automatically refine Platform Independent Models into Platform Specific Models; and *(iii)* rules to transform Platform Specific Models into concrete implementations. We apply our proposal to develop a proof-of-concept example.

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

## 1 Introduction

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

**Computation Independent Models (CIM):** This level focusses on the environment of the system, as well as on its business and requirement specifications. This viewpoint represents the software system at its highest level of abstraction. At this moment of the development, the structure and system processing details are still unknown or undetermined.
**Platform Independent Models (PIM):** This level focusses on the system functionality, hiding the details of any particular platform. The specification defines those parts of the system that do not change from one platform to another.

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

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

In Service-Oriented Computing [**?**], pre-existing services are combined to produce applications and provides the business logic. The selection of services is usually guided by the functional requirements of the application being developed. Some methodologies and techniques have been proposed to help the software developer in the specification of functional requirements of the business logic, such as the Service Oriented Development Method (SOD-M) [**?**]. SOD-M is based on MDD and proposes models, practices and techniques to aid in software development. SOD-M does not provide suport to the specification of non-functional requirements, such as security, reliability, and efficiency. Ideally, non-functional requirements would be considered along with all the stages of the software development. The adoption of non-functional specifications from the early states of development can help the developer to produce applications that are capable of dealing with the application context.

Non-functional properties of service oriented applications have been addressed in academic works and standards [**?**,**?**,**?**]. Dealing with these kind of properties involves the use of specific technologies in different layers of the SOC architecture, for instance during the description of services APIs (such as WSDL[**?**] or REST [**?**]) or to express service coordinations (like WS-BPEL [**?**]).

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

The main goals of our work are: *(i)* To propose a methodology for supporting the construction of service-oriented applications, taking into account both functional and non-functional requirements; *(ii)* To improve the construction process by providing an abstract view of the application and ensure the conformance to its specification; *(iii)* To reduce the programming effort through the semi-automatic generation of models for the application, to produce concrete implementations from high abstraction models;

This paper is organized as follows: Sections **??** and **??** presents, respectively, the SOD-M method of service software process and our proposed extension to deal with non-functional requirements. A proof of concept example is developed in Section **??**. We conclude the paper by presenting some related work and final remarks.

## 2 SOD-M

The Service-Oriented Development Method (SOD-M) [**?**] proposes the usage of the MDD approach in the context of service-based applications. SOD-M provides a framework with models and standards to express functionalities of applications at a high-level of abstraction. SOD-M meta-models are organized into three levels: CIM (*Computational Independent Models*), PIM (*Platform Independent Models*) and PSM (*Platform Specific Models*).

Two models are defined at the CIM level: *value model* and *BPMN model*. The PIM level models the entire structure of the application flow, while, the PSM level provides transformations towards more specific platforms. The PIM-level models are: *use case*, *extended use case*, *service process* and *service composition*. The PSM level models are: *web service interface*, *extended composition service* and *business logic*. These three levels have no support for describing non-functional requirements.

The SOD-M approach includes transformations between models: *CIM-to-PIM, PIM-to-PIM* and *PIM-to-PSM* transformations. Given an abstract model at the CIM level, it is possible to apply transformations for generating a model of the PSM level. In this context, it is necessary to follow the process activities described by the methodology.

SOD-M considers two points of view: *(i) business*, focusing on the characteristics and requirements of the organization, and *(ii) system requirements*, focusing on features and processes to be implemented in order application requirements. In this way, SOD-M aims to simplify the design of service-oriented applications, as well as its implementation using current technologies.

## 3 πSOD-M

πSOD-M provides an environment for building service compositions considering their non-functional requirements. πSOD-M extends the SOD-M meta-models by adding the concept of *Policy* [**?**] to represent non-functional requirements.

πSOD-M proposes the generation of a set of models at different abstraction levels, as well as transformations between these models. πSOD-M models represent both the functional aspects of the application as well as its non-functional constraints. Constraints are restrictions that must be verified during the execution of the application. An example of this is the requirement of the user's authentication for executing some system functions.

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

We extended SOD-M to include non-functional specifications. Our method defines four meta-models: *π-UseCase*, *π-ServiceProcess*, *π-ServiceComposition*

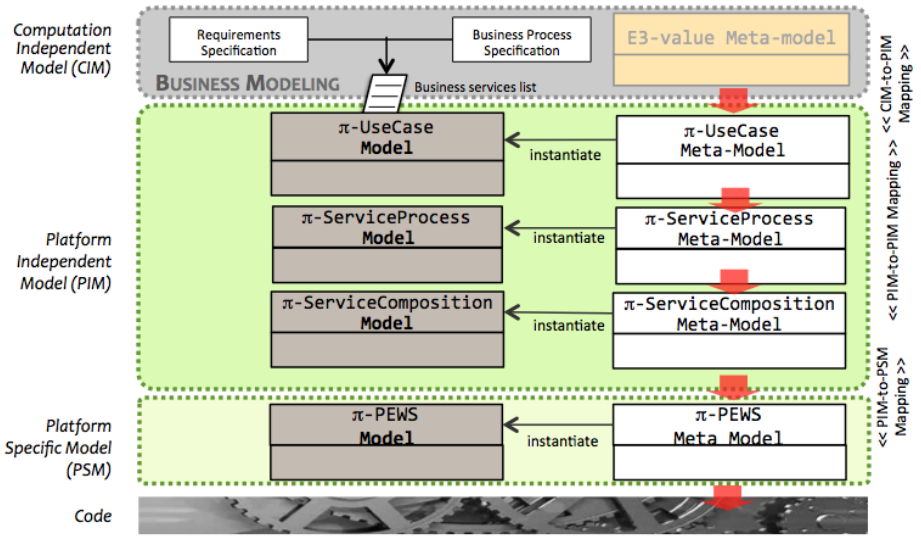



Fig. 1: $\pi$SOD-M.

and $\pi$-*PEWS*. The former three are extensions of SOD-M meta-models and belong to the PIM level. The $\pi$-*PEWS* meta-model is a PSM (Figure **??**).

The $\pi$-*UseCase* meta-model describes functional and non-functional requirements. Non-functional requirements are defined as *constraints* over processing and data. The $\pi$-*ServiceProcess* meta-model defines the concept of *service contract* to represent restrictions over data and actions that must be performed upon certain conditions. The $\pi$-*ServiceProcess* meta-model gathers the constraints described in the $\pi$-*UseCase* model into contracts that are associated with services. The $\pi$-*ServiceComposition* meta-model provides the concept of *Policy* which put together contracts with similar non-functional requirements. For instance, security and privacy restrictions may be grouped into a security policy. $\pi$-*ServiceComposition* models can be refined into PSMs.

At the PSM level we have lower-level models that can be automatically translated into actual computer programs. The $\pi$-*PEWS* meta-model is the PSM adopted in this work. $\pi$-*PEWS* models are textual descriptions of service compositions that can be translated into PEWS code [**?**,**?**]. Although PEWS is our language of choice, other composition languages can be used as target. This can be accomplished by defining: *(i)* a model-to-model transformation, from a $\pi$-*ServiceComposition* model to the corresponding PSM, and *(ii)* a model-to-text transformation, from the this PSM to the composition language.

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

## 4 Proof of Concept: *Tracking Crimes*

Consider a tracking crime application where civilians and police share information about criminality in given zones of a city. Civilian users signal crimes using Twitter. Police officers can notify crimes, as well as update information about solving cases. Some of these information are confidential while other can be shared to the community of users using this application.

Users can track crimes in given zones. Crime information stored by the system may be visualized on a map. Some users have different access rights than others. For example, police officers have more access rights than civilians.

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

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

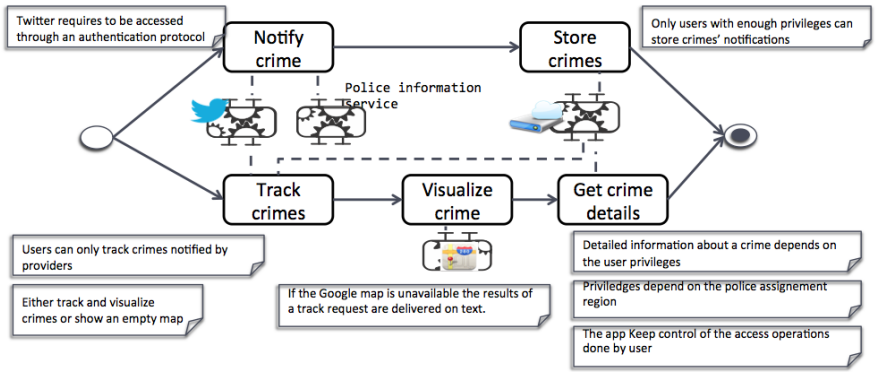



Fig. 2: Business process for the tracking crime example.

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

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

1. Twitter requires to be accessed through an authentication protocol that allows three login failures before blocking.

2. Crime notification needs privileged access.
3. Civilian users can only track crimes for which they have clearance: Civilian population cannot track all the crimes notified by the police.
4. If Google Maps is unavailable, the results are delivered as text.
5. Querying about crimes without having proper clearance yields an empty map.
6. Access rights to detailed information depends on user clearance and zone assignment for police officers.
7. The application maintains a detailed log.

Considering the example of tracking crimes, all the system restrictions are model as constraints in the methodology. We have three types of constraints: *value*, *business* and *exceptions behaviour* constraints. Each use case can be associated to one or more constraints.

*$\pi$-UseCase model:* In our example we have five use cases (Figure **??**), which represent the system functions (tasks) and constraints. We will not detail the functional part of the specification, due to lack of space. The constrains defined for our tracking crime example are:

- The Notify crime task requires that the user is logged in. This is an example of a *value constraint*, where the value associated to the condition depends on the semantics of the application. In this case, it represents the maximmum number of allowed login attempts;
- The Store crimes task requires the verification of the user's clearance (also a value constraint).
- In order to perform the Track crimes task, it is necessary that the notifier user is in the contact list of the requesting user. This is an example of *business constraint*. Additionally the requesting user must be logged in.
- For the View Crime Map task, the specification defines that if the Google Maps service is not available, the result is presented as text. This is an example of *exceptional behaviour constraint*. The availability of the Google Maps service is verified by a *business constraint*.
- The Show crime details task is specified to have three constraints: A *value constraint* is defined to verify the user's clearance level; A *business constraint* is used to ensure that the user's clearance is valid for the geographic zone of the crime; Another *value constraint* defines that the log is to be maintained.

*$\pi$-ServiceProcess model:* The model presented in Figure **??** is transformed, at this stage of the development, into a similar graph, where *(i)* the task nodes are better detailed, by refining the control and data flows; and *(ii)* constraints are transformed into *contracts* (pre- and post-conditions). The new model describes the application's activities and defines contracts for each activity or for parts of the application.

A model-to model transformation is defined in order refine the *$\pi$-UseCase* model of the application into the more detailed model. This (semi-automatic) transformation process is supported by a tool (described in [**?**]).

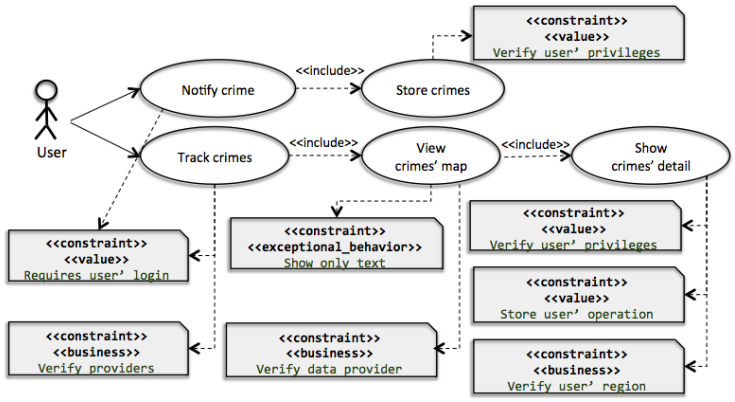



Fig. 3: $\pi$-UseCase Model

The $\pi$-*ServiceProcess* model defined for our tracking crime application is presented in Figure **??**, where:

1. Tasks of the previous model are transformed into *actions*;
2. Actions are grouped into *activities* (in accordance to the business logic).
3. Constraints of the $\pi$-*UseCase* model are transformed into assertions.

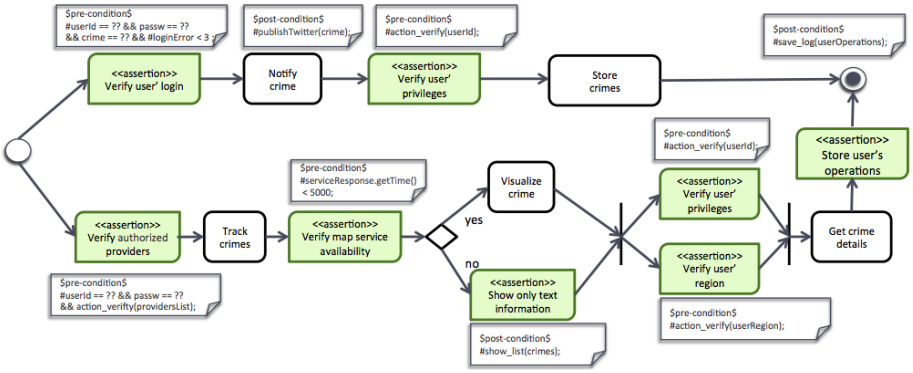
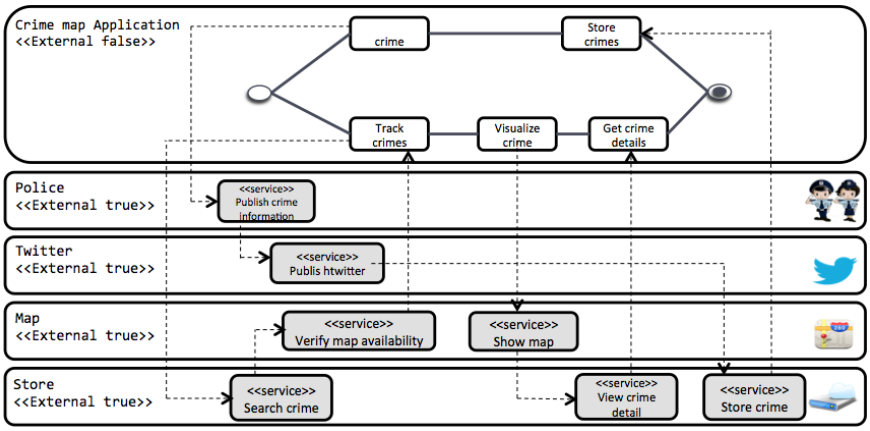
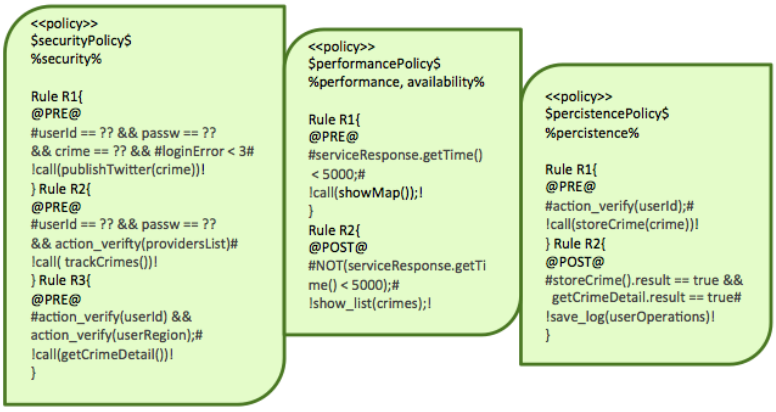



Fig. 4: $\pi$-*ServiceProcess* Model

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




(a) $\pi$-ServiceComposition Model




(b) $\pi$-ServiceComposition Policies

Fig. 5: Service Composition and Policies.

This model describes the service compositions and their related business collaborators (external services), policies, rules associated with a policy and the whole application process and functionalities. The workflow execution will consider the composition of business services and their specific Business Collaborators. Each action will trigger the execution of one or more external services. Before and after each service call, policy rules are verified.

In the case of our crime tracking example, the model produced from the $\pi$-*ServiceProcess* model of Figure **??** is given in Figures **??** and **??**. Figure **??** shows how the crime application interacts with its *busines collaborators* (external services and entities). The interaction occurs by means of function calls (denoted by dotted lines in the figure). Figure **??** shows the definition of three *policies*,

```
//Namespaces specify service URI
1 namespace twitter = www.twitter.com/service.wsdl
2 namespace googlemaps = maps.googleapis.com/maps/api/service
3 namespace amazondynamodb = rds.amazonaws.com/doc/2010-07-28/AmazonRDSv4.wsdl
4 namespace police = www.police.fr/service.wsdl

//Operations
5 alias publishTwitter = portType/publishTwitter in twitter
6 alias searchCrime = portType/searchCrime in amazondynamodb
7 alias showMap = portType/showMap in googlemaps
...
//Services
8  service notifyCrime = publishCrime  . publishTwitter
9  service trackCrime= searchCrime . verifyService
10 Service visualizeCrime =  showMap . getCrimeDetail
...
//Path
11 ( notifyCrime . storeiCrime ) ||
12 ( trackCrime . visualizeCrime . getCrimeDetail )
...
//Contracts
13 defContract notifyCrimeContract{
14 isAppliedTo: notifyCrime
15 requires: userId == ?? && passw == ?? &&
16                                req(notifyCrime)  < 3
17    (OnFailureDo: NOT(action_publish(crime));
18 ensures: publishTwitter(crime) == true
19    (OnFailureDo: skip);
20 }
...
```

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

which define rules for service execution. In our case we have policies for *Security*, *Performance* and *Persistence*.

*$\pi$-PEWS Model:* These models are produced by a model-to-text transformation that takes a *$\pi$-ServiceComposition* model and generates *$\pi$-PEWS* specification code. This code is a service composition program that can be compiled into executable code. *$\pi$-PEWS* models are expressed in a variant of the PEWS composition language.

The $\pi$-PEWS program generated from the model in Figure **??** is partially presented in Figure **??**. The figure shows a simplified program code, produced in accordance to the following guidelines:

1. Namespaces, identifying the addresses of external services are produced from the Business Collaborators of the higher-level model. We define four of them, corresponding to the Police, Twitter, Google Map and Amazon partners.

2. Specific operations exported by each business collaborator are identified to an *operation* of the program (Each operation is given an `alias`).
3. The workflow in Figure **??** is translated into a textual representation (lines 11 and 12 of the program).
4. *Contracts* are defined in $\pi$-PEWS as having pre-conditions (`requires`), post-conditions (`ensures`) and actions (`OnFailureDo`) to be executed in the case thar a condition is not verified. Contracts are generated from Policies (such as those of Figure **??**.

For a more comprehensive account of $\pi$SOD-M the reader can refer to [**?**].

## 5 Related Work and Conclusions

Over the last years, a number of approaches have been proposed for the development of web services. These approaches range from the proposal of new languages for web service descriptions [**?**,**?**] to techniques to support phases of the development cycle of this kind of software [**?**,**?**]. In general, these approaches concentrate on specific problems, like supporting transactions or QoS, in order to improve the security and reliability of service-based applications. Some proposals address service composition: workflow definition [**?**,**?**] or semantic equivalence between services [**?**]. The proposed solutions come from many communities, including those of Theoretical Computer Science [**?**,**?**], Software Engineering [**?**,**?**], Programming Languages [**?**,**?**] and Databases [**?**].

Despite the variety of tecniques proposed, there is not yet a consensus on a software process methodology for web services. Some methodologies address the service-based development towards a standard or a new way to develop reliable applications. SOD-M and SOMF [**?**] are MDD approaches for web services; S-Cube [**?**] is focused on the representation of business processes and service-based development; SOMA [**?**] is a methodology described by IBM for SOA solutions; Extended SOA [**?**] merges RUP and BPM[**?**] concepts for service modeling; DEVISE [**?**] is a methodology for building service-based infrastructure for collaborative enterprises. Other proposals include, the WIED model [**?**], that acts as a bridge between business modeling and design models, and traditional approaches for software engineering [**?**] applied to SOC.

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

$\pi$SOD-M is being used in an academic environment. So far, the preliminary results indicate that $\pi$SOD-M approach is useful for the development of complex web service applications. We are now working on the definition of a PCM-level meta-model to generate BPEL programs (instead of $\pi$-PEWS).