Reliable Web Services Composition: An MDD Approach

Genoveva Vargas-Solar, Valeria de Castro, Placido Antonio Neto Souza, Javier A Espinosa-Oviedo, Esperanza Marcos, Martin A Musicante, José-Luis Zechinelli-Martini, Christine Collet

Reliable Web Services Composition: An MDD Approach

Genoveva Vargas-Solar, Valeria de Castro, Plácido Antonio de Souza Neto, Javier A. Espinosa-Oviedo, Esperanza Marcos, Martin A. Musicante, José-Luis Zechinelli-Martini, and Christine Collet

***Abstract*—This paper presents an approach for modeling and associating *Policies* to services' based appjalications. It proposes to extend the SOD-M model driven method with (i) the π-SCM, a *Policy* services' composition meta-model for representing non-functional constraints associated to services' based applications; (ii) the π-PEWS meta-model providing guidelines for expressing the composition and the policies; and, (iii) model to model and model to text transformation rules for semi-automatizing the implementation of reliable services' compositions. As will be shown within our environment implementing these meta models and rules, one may represent both systems' cross-cutting aspects (e.g., exception handling for describing what to do when a service is not available, recovery, persistence aspects) and constraints associated to services, that must be respected for using them (e.g., the fact that a service requires an authentication protocol for executing a method).**

***Index Terms*—Methodology, πSOD-M, Sevice Composition, Policy**

I. INTRODUCTION

SERVICE oriented computing is at the origin of an evolution in the field of software development. An important challenge of service oriented development is to ensure the alignment between IT systems and the business logic. Thus, organizations are seeking for mechanisms to deal with the gap between the systems developed and business needs [1]. The literature stresses the need for methodologies and techniques for service oriented analysis and design, claiming that they are the cornerstone in the development of meaningful services' based applications [2]. In this context, some authors argue that the convergence of model-driven software development, service orientation and better techniques for documenting and improving business processes are the key to make real the idea of rapid, accurate development of software that serves, rather than dictates, software users' goals [3].

Service oriented development methodologies providing models, best practices, and reference architectures to build services' based applications mainly address functional aspects [4], [5], [6], [7], [8]. Non-functional aspects concerning services' and application's "semantics", often expressed as requirements and constraints in general purpose methodologies, are not fully considered or they are added once the application has been implemented in order to ensure some level of reliability (e.g., data privacy, exception handling, atomicity, data persistence). This leads to services' based applications that are partially specified and that are thereby partially compliant with application requirements.

The objective of this work is to model non-functional constraints and associate them to services' based applications early during the services' composition modeling phase. Therefore this paper presents πSOD-M, a model-driven method that extends the SOD-M [6] for building reliable services' based information systems (SIS).

This work (i) proposes to extend the SOD-M [6] method with the notion of *A-Policy* [9] for representing non-functional constraints associated to services' based applications; (ii) defines the π-PEWS meta-model providing guidelines for expressing the composition and the *A-policies*; and finally, (iii) defines model to model transformation rules for generating the π-PEWS model of a reliable services' composition starting from the extended services' composition model; and, model to text transformations for generating the corresponding implementation. As will be shown within our environment implementing these meta models and rules, one may represent both systems' cross-cutting aspects (e.g., exception handling for describing what to do when a service is not available, recovery, persistence aspects) and constraints associated to services, that must be respected for using them (e.g., the fact that a service requires an authentication protocol for executing a method).

Manuscript received on April 24, 2012; accepted for publication on June 12, 2014.

Genoveva Vargas-Solar is with French Council of Scientific Research, LIG-LAFMIA, 681 rue de la Passerelle BP 72, 38402 Saint Martin d'Hères, France (e-mail: Genoveva.Vargas@imag.fr).

Valeria de Castro is with Universidad Rey Juan Carlos, Av Tulipán, Móstoles, Spain (e-mail: Valeria.deCastro@urjc.es).

Plácido Antonio de Souza Neto is with Instituto Federal do Rio Grande do Norte, Av. Senador Salgador Filho, 1559 – Tirol, Natal – RN, Brasil (e-mail: placido.neto@ifrn.edu.br).

Javier A. Espinosa-Oviedo is with French Mexican Laboratory of Informatics and Automatic Control, 681 rue de la Passerelle BP 72, 38402 Saint Martin d'Hères, France (e-mail: javiera.espinosa@gmail.com).

Esperanza Marcos is with Universidad Rey Juan Carlos, Av Tulipán, Móstoles, Spain (e-mail: esperanza.marcos@urjc.es).

Martin A. Musicante is with DIMAp - UFRN, ForAll - Formal Methods and Language Research Laboratory Campus Universitrio – Lagoa Nova, Natal – RN, Brasil (e-mail: mam@dimap.ufrn.br).

José-Luis Zechinelli-Martini is with French-Mexican Laboratory on Informatics and Automatic Control, 681 rue de la Passerelle BP 72, 38402 Saint Martin d'Héres, France (e-mail: joseluis.zechinelli@gmail.com).

Christine Collet is with Grenoble Institute of Technology, Laboratory of Informatics of Grenoble, 681 rue de la Passerelle BP 72, 38402 Saint Martin d'Hères, France (e-mail: Christine.Collet@imag.fr).

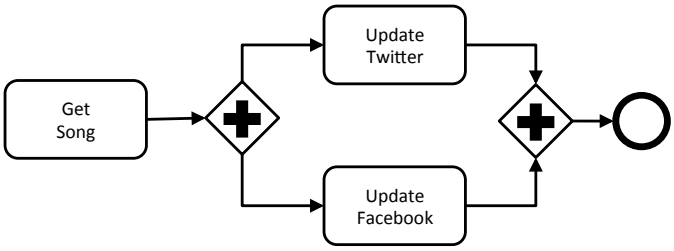

Fig. 1. BPMN model of the "To Publish Music" scenario

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

II. MODELING RELIABLE SERVICES' COMPOSITIONS WITH πSOD-M

Consider for instance the following scenario. An organization wants to provide the services' based application "To Publish Music" that monitors the music a person is listening during some periods of time and sends the song title to this person's Twitter and Facebook accounts. Thus, this social network 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. For developing this services' based application it is necessary to compose the following services calling their exported methods:

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

Figure 1 shows the BPMN model[1] of the scenario. The "To Publish Music" scenario 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).

Given a set of services with their exported methods known in advance or provided by a service directory, building services' based applications can be a simple task that implies expressing an application logic as a services' composition. The challenge being ensuring the compliance between the specification and the resulting application. Software engineering methods (e.g., [4], [5], [6], [7]) today can help to ensure this compliance, particularly when information systems include several sometimes complex business processes calling Web services or legacy applications exported as services.

A. Modeling a Services' Based Application

Figure 2 shows SOD-M that defines a service oriented approach providing a set of guidelines to build services' based information systems (SIS) [6], [10]. Therefore, SOD-M proposes to use services as first-class objects for the whole process of the SIS development and it follows a Model Driven Architecture (MDA) [11] approach. Extending from the highest level of abstraction of the MDA, SOD-M provides a conceptual structure to: first, capture the system requirements and specification in high-level abstraction models (computation independent models, CIM's); next, starting from such models build platform independent models (PIM's) specifying the system details; next transform such models into platform specific models (PSM's) that bundles the specification of the system with the details of the targeted platform; and finally, serialize such model into the working-code that implements the system.

As shown in Figure 2, the SOD-M model-driven process begins by building the high-level computational independent models and enables specific models for a service platform to be obtained as a result [6]. Referring to the "To Publish Music" application, using SOD-M the designer starts defining an E3value model[2] at the CIM level and then the corresponding models of the PIM are generated leading to a services' composition model (SCM).

Now, consider that besides the services' composition that represents the order in which the services are called for implementing the application "To Publish Music" it is necessary to model other requirements that represent the (i) conditions imposed by services for being contacted, for example the fact the Facebook and Twitter require authentication protocol in order to call their methods for updating the wall; (ii) the conditions stemming from the business rules of the application logic, for example 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.

B. Modeling Non-functional Constraints of Services' Based Applications

Adding non-functional requirements and services constraints in the services' composition is a complex task that

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

[2]The E3 value model is a business model that represents a business case and allows to understand the environment in which the services' composition will be placed [12].

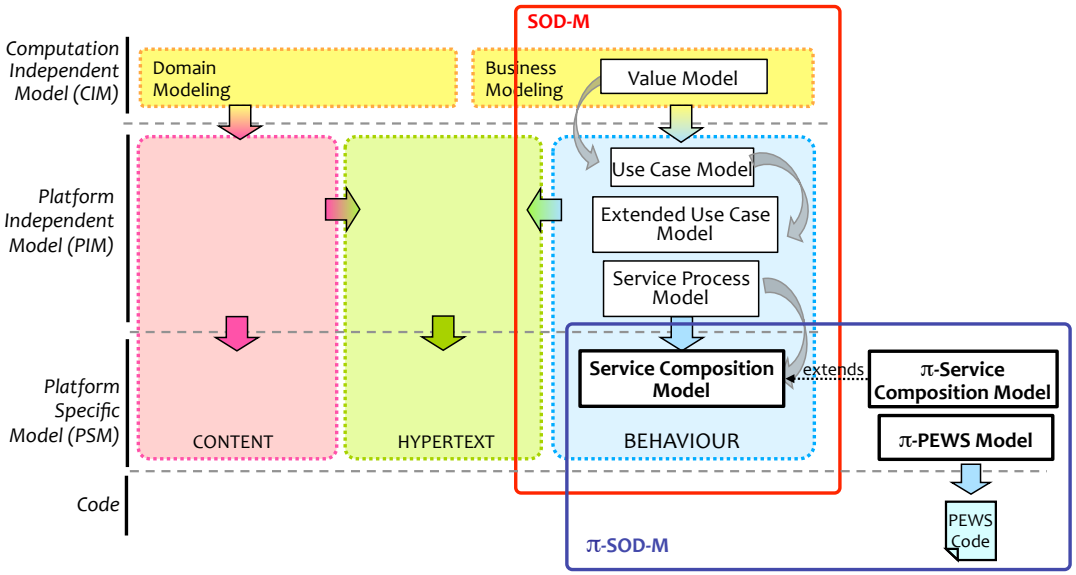

Fig. 2. SOD-M development process

implies programming protocols for instance authentication protocols to call Facebook and Twitter in our example, and atomicity (exception handling and recovery) for ensuring a true synchronization of the song title disseminated in the walls of the user's Facebook and Twitter accounts.

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

This work extends SOD-M for building applications by modeling the application logic and its associated non-functional constraints and thereby ensuring the generation of reliable services' composition. As a first step in our approach, and for the sake of simplicity we started modeling non-functional constraints at the PSM level. Thus, in this paper we propose the π-SCM, the services' composition meta-model extended with *A-policies* for modeling non-functional constraints (highlighted in Figure 2 and described in Section III). πSOD-M defines the π-PEWS meta-model providing guidelines for expressing the services' composition and the *A-policies* (see Section IV), and also defines model to model transformation rules for generating π-PEWS models starting from π-SCM models that will support executable code generation (see Section V). Finally, our work defines model to text transformation rules for generating the program that implements both the services' composition and the associated *A-policies* and that is executed by an adapted engine (see Section VI).

III. π-SERVICES' COMPOSITION META-MODEL

The *A-policy* based services' composition meta-model (see in Figure 4) represents a workflow needed to implement a services' composition, identifying those entities that collaborate in the business processes (called BUSINESS COLLABORATORS[3]) and the ACTIONS that they perform. This model is represented by means of a UML activity diagram. Thus, as shown in Figure 3, the meta-model includes typical modeling elements of the activity diagram such as ACTIVITYNODES, INITIALNODES and FINALNODES, DECISIONNODES, etc., along with new elements defined by SOD-M such as BUSINESS COLLABORATORS, SERVICEACTIVITY and ACTION (see the white elements in Figure 4).

- A BUSINESS COLLABORATOR element represents those entities that collaborate in the business processes by performing some of the required actions. They are graphically presented as a partition in the activity diagram. A collaborator can be either internal or external to the system being modelled. When the collaborator of the business is external to the system, the attribute IsExternal[4] of the collaborator is set to true.
- ACTION, a kind of EXECUTABLENODE, are represented in the model as an activity. Each action identified in

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

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

the model describes a fundamental behaviour unit which represents some type of transformation or processing in the system being modelled. There are two types of actions: i) a WebService (attribute Type is WS); and ii) a simple operation that is not supported by a Web Service, called an ACTIVITYOPERATION (attribute Type is AOP).
– The SERVICEACTIVITY element is a composed activity that must be carried out as part of a business service and is composed of one or more executable nodes.

To illustrate the use of the π-SCM meta-model we used it for defining the *A-policy* based composition model of the "To Publish Music" scenario (see Figure 4). There are three external business collaborators (*Spotify, Twitter* and *Facebook*[5]). It also shows the business process of the "To Publish Music" application that consists of three service activities: *Listen Music*, *Public Music* and *Confirmation*. Note that the action *Publish Music* of the application calls the actions of two service collaborators namely *Facebook* and *Twitter*.

Instead of programming different protocols within the application logic, we propose to include the modeling of non-functional constraints like transactional behaviour, security and adaptability at the early stages of the services' composition engineering process. We model non-functional constraints of services' compositions using the notion of *A-policy* [9], [13], a kind of pattern for specifying *A-policy* types. In order to represent constraints associated to services compositions, we extended the SOD-M services' composition model with two concepts: RULE and A-POLICY (see blue elements in the π-SCM meta-model in Figure 3).

The RULE element represents an event - condition - action rule where the EVENT part represents the moment in which a constraint can be evaluated according to a condition represented by the CONDITION part and the action to be executed for reinforcing it represented by the ACTION part. An *A-policy* groups a set of rules. It describes global variables and operations that can be shared by the rules and that can be used for expressing their Event and Condition parts. An *A-Policy* is associated to the elements BUSINESSCOLLABORATOR, SERVICEACTIVITY and, ACTION of the π-SCM meta-model (see Figure 3).

Given that *Facebook* and *Twitter* services require authentication protocols in order to execute methods that will read and update the users' space. A call to such services must be part of the authentication protocol required by these services. In the example we associate two authentication policies, one for the open authentication protocol, represented by the class Twitter OAuthPolicy that will be associated to the activity UpdateTwitter (see Figure 4). In the same way, the class Facebook HTTPAuthPolicy, for the http authentication protocol will be associated to the activity UpdateFacebook. OAuth implements the open authentication protocol. As shown in Figure 4, the *A-policy* has a variable Token that will be used to store the authentication token provided by the service. This variable type is imported through the library OAuth.Token. The *A-policy* defines two rules, both can be triggered by events of type ActivityPrepared: (i) if no token has been associated to the variable token, stated in by the condition of rule R_1, then a token is obtained (action part of R_1); (ii) if the token has expired, stated in the condition of rule R_2, then it is renewed (action part of R_2). Note that the code in the actions profits from the imported OAuth.Token for transparently obtaining or renewing a token from a third party.

HTTP-Auth implements the HTTP-Auth protocol. As shown in Figure 4, the *A-policy* imports an http protocol library and it has two variables username and password. The event of type ActivityPrepared is the triggering event of the rule R_1. On the notification of an event of that type, a credential is obtained using the username and password values. The object storing the credential is associated to the scope, i.e., the activity that will then use it for executing the method call.

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

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

IV. π-PEWS META-MODEL

The idea of the π-PEWS meta-model is based on the services' composition approach provided by the language PEWS [14], [15] (*Path Expressions for Web Services*), a programming language that lets the service designer combine the methods or subprograms that implement each operation of a service, in order to achieve the desired application logic. Figure 5 presents the π-PEWS meta-model consisting of classes representing:

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

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

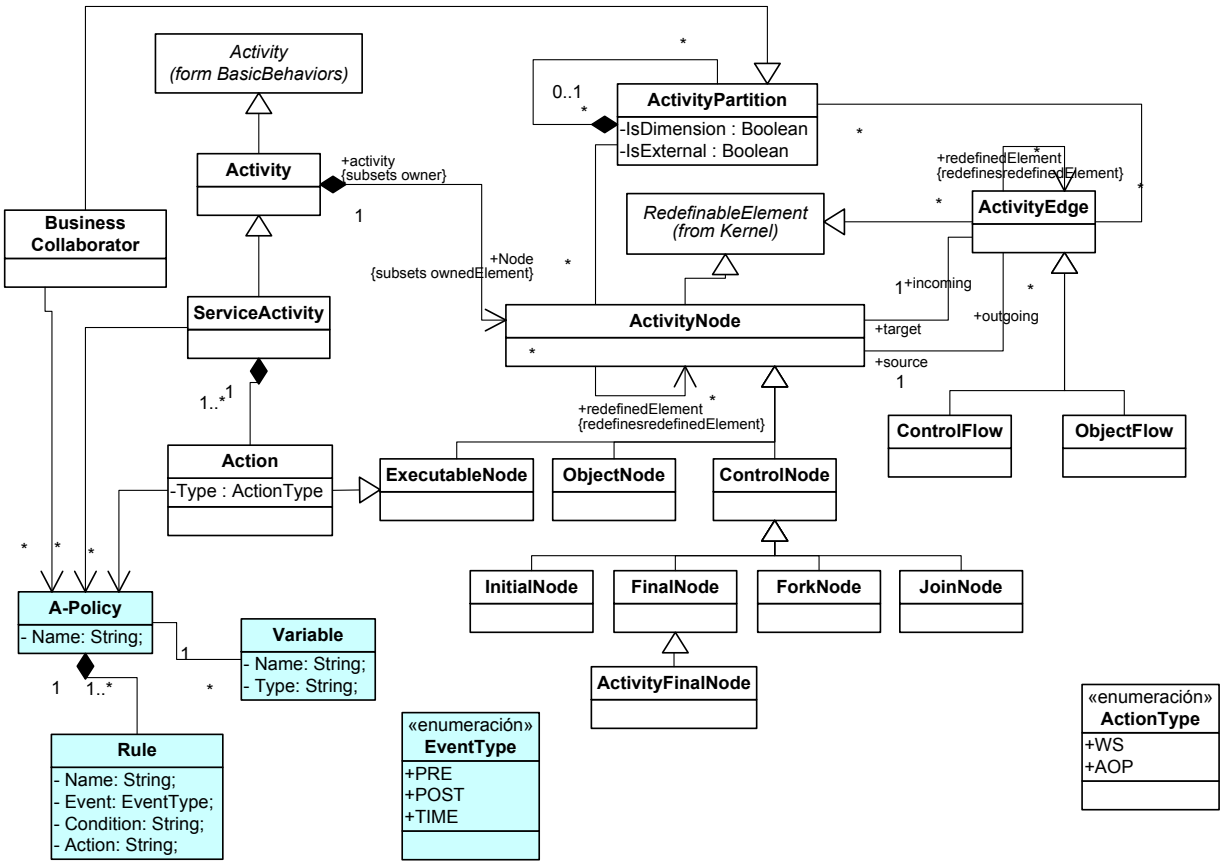

Fig. 3. *A-policy* based services' composition meta-model (π-SCM)

operation or the conditional execution of an operation ($[C]S$).

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

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

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

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

V. TRANSFORMATION RULES

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

A. Transformation of the Services' Composition Elements of the π-SCM to the π-PEWS Elements

A named action of the π-SCM represented by *Action* and *Action:name* is transformed to a named class OPERATION

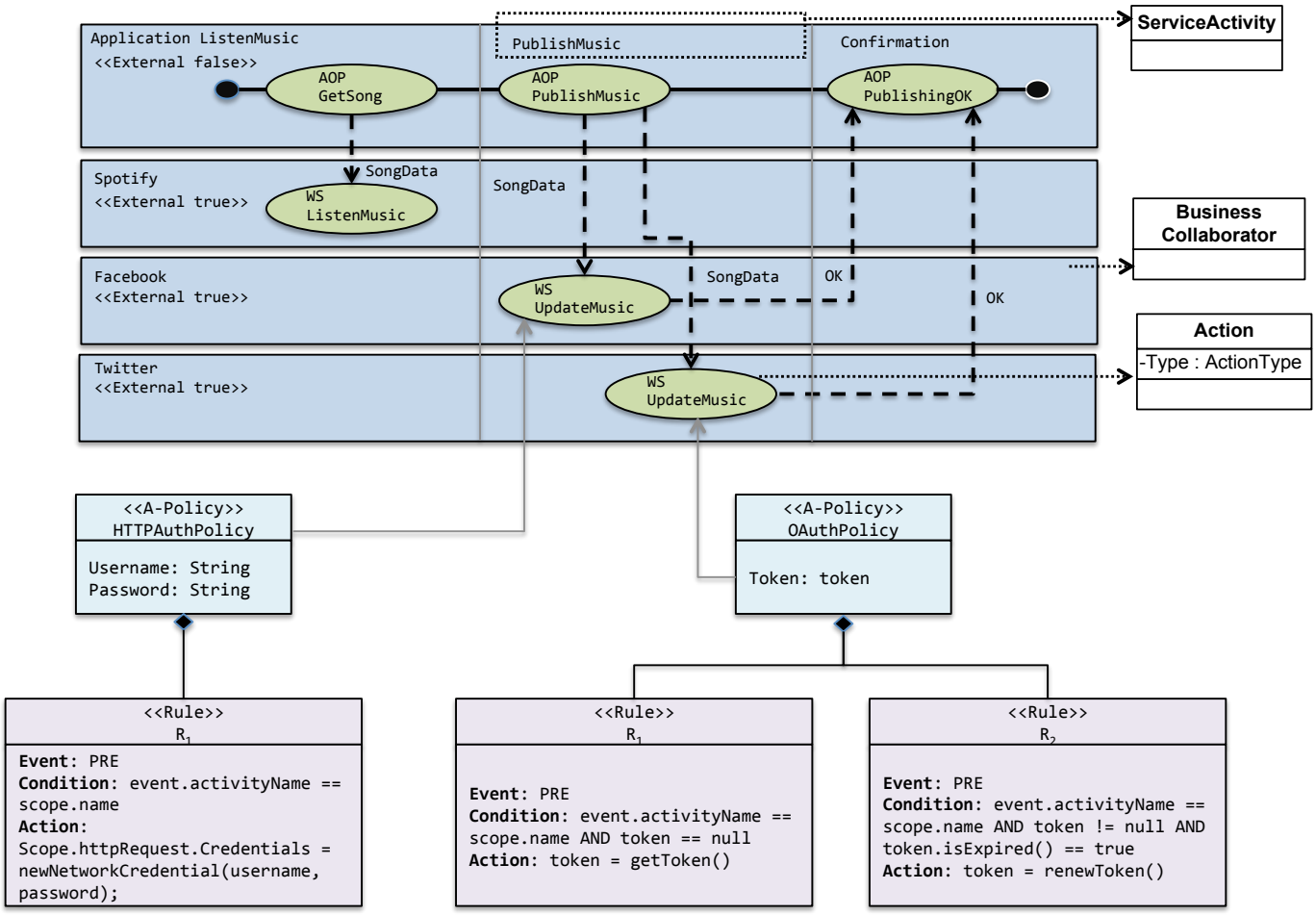

Fig. 4. Services' composition model for the business service "To publish music"

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

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

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

B. Transformation of Rules Grouped by A-policies in the π-SCM to A-Policies of π-PEWS

The *A-policies* defined for the elements of the π-SCM are transformed into A-POLICY classes, named according to the names expressed in the source model. The transformation of the rules expressed in the π-SCM is guided by the event types associated to these rules. The variables associated to an *A-policy* expressed in the π-SCM as $<$*Variable:name, Variable:type*$>$ are transformed into elements of type VARIABLE with attributes NAME and TYPE directly specified from the elements *Variable:name* and *Variable:type* of the π-SCM model.

As shown in Table I, for an event of type *Pre* the corresponding transformed rule is of type PRECONDITION; for an event of type *Post* the corresponding transformed rule is of type POSTCONDITION; finally, for an event of type *TimeRestriction* the corresponding transformed rule

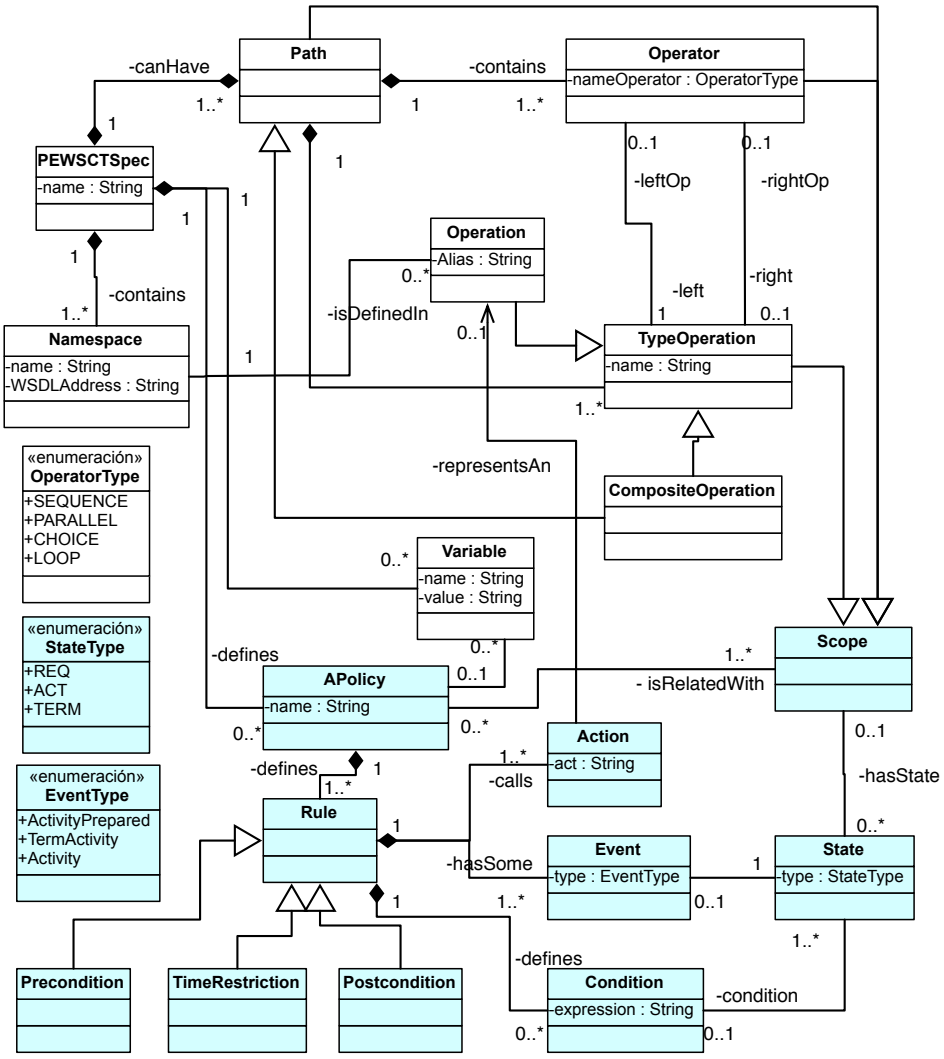

Fig. 5. π-PEWS Metamodel

is of type TIME. The condition expression of a rule in the π-SCM (*Rule:condition*) is transformed into a class *Condition:expression* where the attributes of the expression are transformed into elements of type ATTRIBUTE.

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

VI. IMPLEMENTATION ISSUES

This section describes the πSOD-M development environment that implements the generation of *A-policies*' based services' compositions. For a given services' based application, the process consists in generating the code starting from a π-SCM modeling an application. Note that the services' composition model is not modeled from scratch, but it is the result of a general process defined by the πSOD-M method in which a set of models are built following a service oriented approach [6].

A. πSOD-M Development Environment

Figure 6 depicts a general architecture of the πSOD-M Development Environment showing the set of plug-ins developed in order to implement it. The environment implements the abstract architecture shown in Figure 2. Thus, it consists of plug-ins implementing the π-SCM and π-PEWS meta-models used for defining models specifying services' compositions and their associated policies; and ATL rules for transforming PSM models (model to model transformation) and finally generating code (model to text transformation).

TABLE I
TRANSFORMATION RULES: FROM π-SERVICECOMPOSITION TO π-PEWS

Source: π-SCM	Mapping Rules	Target: π-PEWS
Action	– An *Action* in the source model corresponding to an external **Business Collaborator** is mapped to an Operation in target model. – The **Action:name** in the source model is transformed into **Operation:name** in the target model.	Operation: alias
Service Activity	– The **ServiceActivity** in the source model is mapped to a **Composite Operation** in target model when more than one **Actions** are called. – If **Composite Operation** is generated for a given **Service Activity** then the **ServiceActivity:name** in the source model is mapped to **CompositeOperation:name** in the target model.	Type Operation, Composite Operation
Control Nodes	– The **Control Node** in the source model is mapped to a **Operator** in target model. According to the type of **Control Node** (merge, decision, join, fork) the expression of the **Composite Operation** is: • Sequence if no **ControlNode** is specified; • Parallel - Sequence for a **ControlNodes** pattern fork – join; • Choice - Sequence for a **ControlNodes** pattern decision – merge	Operator
Business Collaborator	A **BusinessCollabortor:isExternal** in the source model generates a **Namespaces** in the target model	Namespace
Rule:event	The **Rule**'s attribute event in the source model is transformed into an **Event:type** of the target model. In this case attribute is mapped to an entity with an attribute. The **Event Type** of a **Rule** in the target model is determined by the Rule type: • **Event Type** of a *Precondition* **Rule** is *ActivityPrepared*; • **Event Type** of a *Postcondition* **Rule** is *TermActivity*; • **Event Type** of a *TimeRestriction* **Rule** is *Activity*	Event Type, Event
Rule: condition	The **Rule**'s attribute condition in the source model is transformed into a **Condition:expression** in the target model. In this case, an attribute is mapped into an entity with an attribute.	Condition
Rule:action	The **Rule:action** in the source model is transformed in an **Action:act** in the target model. The attribute action is mapped to an entity with an attribute. In the target model an action is executed according to the rule condition value (true/false).	Action
Policy	– Every **Policy** associated to an element (**Business Collaborator**, **Service**, **Activity**, **Action**) in the source model becomes an **APolicy** associated to the corresponding element in the target model. – The name attribute of a **Policy** in the source model becomes an **Apolicy:name** of the target model.	APolicy
Variable	Every **Variable**, and its attributes, associated to a **Policy** in the source model becomes a **Variable** associated to an **APolicy** in the target model. The variables can be used in an **APolicy**'s Condition of the target model.	Variable
Rule:event	For a **Rule** in the source model, depending on the **Event Type**, the corresponding transformation in the target model is: **Precondition, Postcondition** or **Time Restriction Rule**	Precondition, Postcondition, Time Restriction, Rule

– We used the Eclipse Modeling Framework (EMF)[6] to implement the meta-models π-SCM and π-PEWS. Starting form these meta-models, we developed the models' plug-ins needed to support the graphical representation of the π-SCM and π-PEWS models (π-ServiceCompostion Model and π-PEWS Model plug-ins).
– We used ATL[7] for developing the mapping plug-in implementing the mappings between models (π-ServiceComposition2π-PEWS Plug-in).
– We used Acceleo[8] for implementing the code generation plug-in. We coded the pews.mt program that implements the model to text transformation for generating executable code. It takes as input a π-PEWS model implementing a specific services' composition and it generates the code to be executed by the *A-policy* based services' composition execution environment.

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

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

At execution time, the evaluation of policies done by the *A-policy* manager must be synchronized with the execution of the services' composition (i.e., the execution of an activity or a control flow). Policies associated to a scope are activated when the execution of its scope starts. A *A-policy* will have to be executed only if one or several of its rules is triggered. If

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

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

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

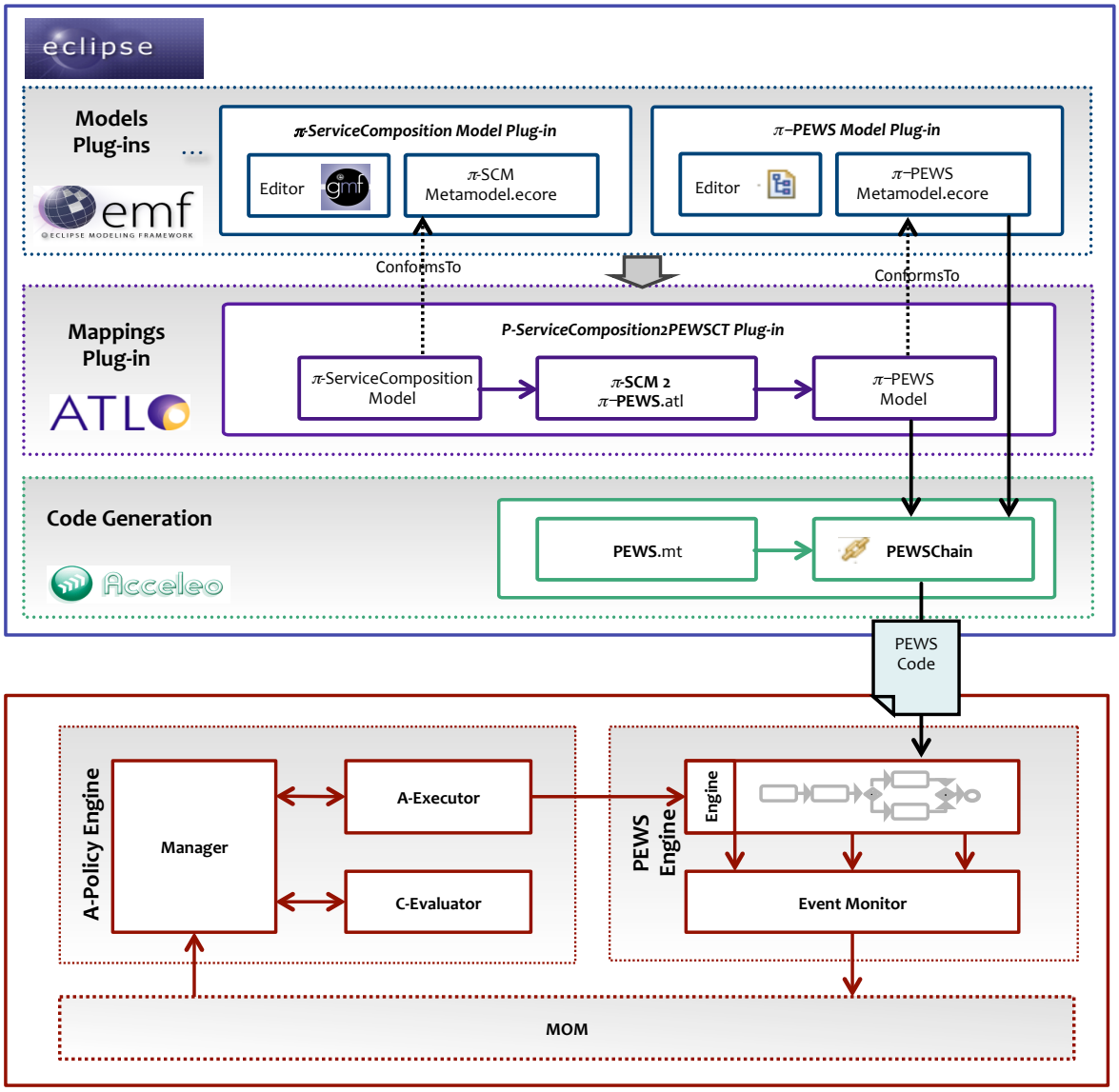

Fig. 6. πSOD-M Development Environment

several rules are triggered the *A-policy* manager first builds an execution plan that specifies the order in which such rules will be executed according to the strategies defined in the following section. If rules belonging to several policies are triggered then policies are also ordered according to an execution plan. The execution of policies is out of the scope of this paper, the interested reader can refer to [9] for further details.

VII. Related Work

Work related with our approach includes standards devoted for expressing non-functional constraints for services and services' compositions. It also includes methods and approaches for modeling non-functional constraints.

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

The selection of the adequate protocols for adding a specific non-functional constraints to a services' composition (e.g., security, transactional behaviour and adaptability) is responsibility of a programmer. As a consequence, the development of an application based on a services' composition is a complex and a time-consuming process. This is opposed to the philosophy of services that aims at facilitating the integration of distributed applications. Other works, like [16] introduce a model for transactional services

composition based on an advanced transactional model. [17] proposes an approach that consists of a set of algorithms and rules to assist designers to compose transactional services. In [18] the model introduced in [19] is extended to web services for addressing atomicity.

There are few methodologies and approaches that address the explicit modeling of non functional properties for service based applications. Software process methodologies for building services based applications have been proposed in [7], [20], [21], [22], and they focus mainly on the modeling and construction process of services based business processes that represent the application logic of information systems.

Design by Contract [23] is an approach for specifying web services and verifying them through runtime checkers before they are deployed. A contract adds behavioral information to a service specification, that is, it specifies the conditions in which methods exported by a service can be called. Contracts are expressed using the language *jmlrac* [24]. The *Contract Definition Language* (CDL) [20] is a XML-based description language, for defining contracts for services. There are an associated architecture framework, design standards and a methodology, for developing applications using services. A services' based application specification is generated after several [25] B-machines refinements that describe the services and their compositions. [7] proposes a methodology based on a SOA extension. This work defines a service oriented business process development methodology with phases for business process development. The whole life-cycle is based on six phases: planning, analysis and design, construction and testing, provisioning, deployment, and execution and monitoring.

VIII. Conclusions and Future Work

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

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

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

Acknowledgments

This work was partially financed by the projects CLEVER, STIC-AMSUD, and MASAI. P. A. de Souza Neto was funded by CAPES/STIC-AMSUD Brasil, BEX 4112/11-3.

References

[1] M. Bell, *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley, New Jersey, 2008.

[2] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer*, vol. 40, no. 11, 2007.

[3] A. Watson, "A brief history of MDA," 2008.

[4] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley, "SOMA: A method for developing service-oriented solutions," *IBM System Journal*, vol. 47, no. 3, 2008.

[5] A. W. Brown, S. K. Johnston, G. Larsen, and J. Palistrant, "SOA Development Using the IBM Rational Software Development Platform: A Practical Guide," in *Rational Software*, 2005.

[6] 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*, vol. 18, no. 2, 2009.

[7] M. P. Papazoglou and W.-J. van den Heuvel, "Service-oriented design and development methodology," *Int. J. Web Eng. Technol.*, vol. 2, no. 4, pp. 412–442, 2006.

[8] P. Queiroz and R. Braga, "Application engineering of service-based software product lines," in *SAC*, 2012, pp. 1996–1997.

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

[10] V. De Castro, E. Marcos, and J. Vara, "Applying cim-to-pim model transformations for the service-oriented development of information systems," *IInformation and Software Technology*, vol. 53, no. 19, 2011.

[11] J. Miller and J. Mukerji, "MDA guide," 2003, downloaded on 27-Jun-2014. [Online]. Available: http://www.omg.org/cgi-bin/doc?omg/03-06-01

[12] J. Gordijn and J. Akkermans, "Value based requirements engineering: Exploring innovative e-commerce idea," *Requirements Engineering Journal*, vol. 8, no. 2, 2003.

[13] J. A. Espinosa-Oviedo, G. Vargas-Solar, J.-L. Zechinelli-Martini, and C. 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: ACM, 2009.

[14] C. Ba, M. Halfeld-Ferrari, and M. A. Musicante, "Composing web services with PEWS: A trace-theoretical approach," in *ECOWS*, 2006, pp. 65–74.

[15] P. A. Souza Neto, M. A. Musicante, G. Vargas-Solar, and J.-L. Zechinelli-Martini, "Adding contracts to a web service composition language," *LTPD – 4th Workshop on Languages and Tools for Multithreaded, Parallel and Distributed Programming*, September 2010.

[16] M.-C. Fauvet, H. Duarte, M. Dumas, and B. Benatallah, "Handling transactional properties in web service composition," in *WISE 2005: 6th International Conference on Web Information Systems Engineering*, vol. 3806. LNCS, Springer-Verlag, Octobre 2005, pp. 273–289.

[17] S. Bhiri, C. Godart, and O. Perrin, "Reliable web services composition using a transactional approach," in *e-Technology, e-Commerce and e-Service*, ser. EEE, vol. 1. IEEE, March 2005, pp. 15–21.

[18] K. Vidyasankar and G. Vossen, "A multi-level model for web service composition," in *ICWS*. IEEE Computer Society, 2004, p. 462.

[19] H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek, "Atomicity and Isolation for Transactional Processes," *ACM Transactions on Database Systems (TODS)*, vol. 27, no. 1, pp. 63–116, Mar. 2002.

[20] N. Milanovic, "Contract-based web service composition," Ph.D. dissertation, Humboldt-Universität zu Berlin, 2006.

[21] G. Feuerlicht and S. Meesathit, "Towards software development methodology for web services," in *SoMeT*, 2005, pp. 263–277.

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

[23] R. Heckel and M. Lohmann, "Towards contract-based testing of web services," in *Proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS 2004)*, M. Pezzé, Ed., vol. 116, 2005, pp. 145–156. [Online]. Available: http://www.cs.le.ac.uk/people/rh122/papers/2005/HL05TACoS.pdf

[24] G. T. Leavens, Y. Cheon, C. Clifton, C. Ruby, and D. R. Cok, "How the design of JML accomodates both runtime assertion checking and formal verification," in *FMCO*, 2002, pp. 262–284.

[25] J.-R. Abrial, M. K. O. Lee, D. Neilson, P. N. Scharbach, and I. H. Sørensen, "The B-method," in *VDM Europe (2)*, ser. Lecture Notes in Computer Science, vol. 552. Springer, 1991, pp. 398–405.