WORKPACKAGE 2: METHOD ENGINEERING

D2.4

DEFINITION OF THE METHOD UNDERLYING SOFTWARE ARCHITECTURE

Project acronym: UsiXML

Project full title: User interface eXtensible Mark-up Language

ITEA label n°08026

DOCUMENT CONTROL

***Deliverable N°**: D2.4*
***Due Date**: 10/2011*
***Delivery Date**: 10/2011*

***Short Description**:* Task 2.4 aims to define a software architecture for enacting a UI development method based on the principles of Service Oriented Architecture (SOA).The architecture supports the following process: the method is specified by the SPEM4UsiXML meta-model (proposed in Task 2.1); the SPEM4UsiXML definition is transformed into a BPEL process in order to describe the method enactment; the BPEL process result is then processed by a BPEL engine that invokes the different UsiXML model transformation services via a SOAP protocol.

***Lead Partner**: UND*
***Contributors:** UND*
***Made available to**:*

Rev	Date	Author	Checked by	Internal Approval	Description
1.0	30/09/11	Draft – Mohamed Boukhebouze, Philippe Thiran, UND.			Initial version
1.1	27/04/12	Draft – Mohamed Boukhebouze, Philippe Thiran, UND.	D. Faure, THA		Reviewed after the review of Nathalie Aquino (UPV)

CONTENTS

1. EXECUTIVE SUMMARY

Task 2.4 aims to define a software architecture that supports the UsiXML methods. This architecture allows the definition and the enactment of the UsiXML methods based on SPEM4UsiXML, the meta-model proposed in Task 2.1. This meta-model relies on the OMG standard SPEM 2.0 meta-model, which uses a UML profile to define the elements of a method. SPEM4UsiXML allows expressing the core elements of the UsiXML methods (like the development path, the development step, and the development sub-step). In addition, the meta-model separates the static aspect of a UsiXML method (the method content), from the dynamic aspect of a method (the process structure). Like in SPEM, there is a lack of support for method enactments in SPEM4UsiXML. To deal with this limitation, we propose an architecture that supports the enactment of the UsiXML methods. The enactment consists in transforming a SPEM4UsiXML model to a BPEL model so that it can be executed by any BPEL engine.

2. DOCUMENTS

1.1. Reference

D2.4 UsiXML	

3. INTRODUCTION

A UsiXML method represents a process that stepwise transforms UsiXML models in order to obtain specifications that are detailed and precise enough to be rendered or transformed into code [4]. A UsiXML method is also used to synthesize abstract models from detailed models. To achieve the UsiXML transformations, different types of transformation mechanisms can be used like the reification, the abstraction and the code generation Beuvens These different UsiXML transformation types are instantiated by development steps [4]. These development steps may be combined to form a UsiXML method. The process of combining development steps into a UsiXML method is called a development path. Vanderdonckt et al., in [4], identify several types of development paths, like the forward engineering, the reverse engineering and context of use adaptation.

To reap all the benefits, a UsiXML method needs to formally describe its content (its semantics) and its form (its abstract/concrete syntax). For this reason, a UsiXML method needs to be compliant with a well-defined meta-model so that the core elements of UsiXML methods (e.g., the development path, the development step and the development sub-step) can be formally defined. In addition, the enactment of the UsiXML methods needs to be supported by a tool. By enactment of a UsiXML method, we mean the ability of a tool to support UsiXML model transformations according to the method specification.

In this document, we propose a software architecture that supports UsiXML methods. This architecture allows the definition and the enactment of UsiXML methods. The definition of a UsiXML method (in this architecture) is based on the SPEM4UsiXML meta-model that is proposed as a deliverable of the UsiXML project, Task 2.1 [7]. This meta-model extends SPEM

2.0 ([5]), a standard from OMG, by adding new classes that support the specific key elements of the UsiXML methods (e.g. development path, development step and development sub-step). So that, SPEM4UsiXML inherits from SPEM a great usability since it is a UML profile. Unfortunately, like SPEM, the SPEM4UsiXML meta-model cannot support the enactment of a UsiXML method on a specific endeavor. Indeed, the SPEM4UsiXML meta-model allows the description of a method process structure without introducing its own formalism to precisely describe the process behavior models. [5] argues that the separation of the SPEM method structure from the behavior of the method opens up the possibility to reuse existing externally-defined behavior models. A method described with the SPEM 2.0 meta-model can be enacted by mapping it to a business flow or an execution language such as BPEL [1] or XPDL [10] and then executing this representation of processes using enactment engines such as a BPEL engine [5]. Our software architecture for supporting the UsiXML methods allows the enactment of a UsiXML method by transforming a SPEM4UsiXML model to a BPEL specification based on a set of mapping rules and by executing the BPEL specification using a BPEL engine.

The rest of the document gives an overview of the software architecture that supports UsiXML methods by presenting the definition, the deployment and the enactment of UsiXML methods.

4. SOFTWARE ARCHITECTURE FOR SUPPORTING USIXML METHODS

In this section, we describe our proposed software architecture that allows the definition and the enactment of UsiXML methods. The architecture is based on the principles of Service Oriented Architecture (SOA) since a UsiXML method can be seen as a Web services composition. Indeed, in our proposed architecture, UsiXML model transformations are implemented as Web services. Each Web service enacts a specific development sub-step by using associated transformation rules so that, the UsiXML model transformation is more flexible and independent to any transformation system. To achieve this objective, the proposed architecture, depicted in Figure 1, supports the following processes: a UsiXML method is defined, at design-time, based on the SPEM4UsiXML meta-model; the SPEM4UsiXML definition is then transformed, at deployment-time, into a BPEL process in order to describe the method enactment; the BPEL process result is then processed, at runtime, by a BPEL engine that invokes the different transformation Web services via a SOAP protocol; finally, the method enactment is controlled, at diagnostic-time, so that problems are detected whenever they occur.

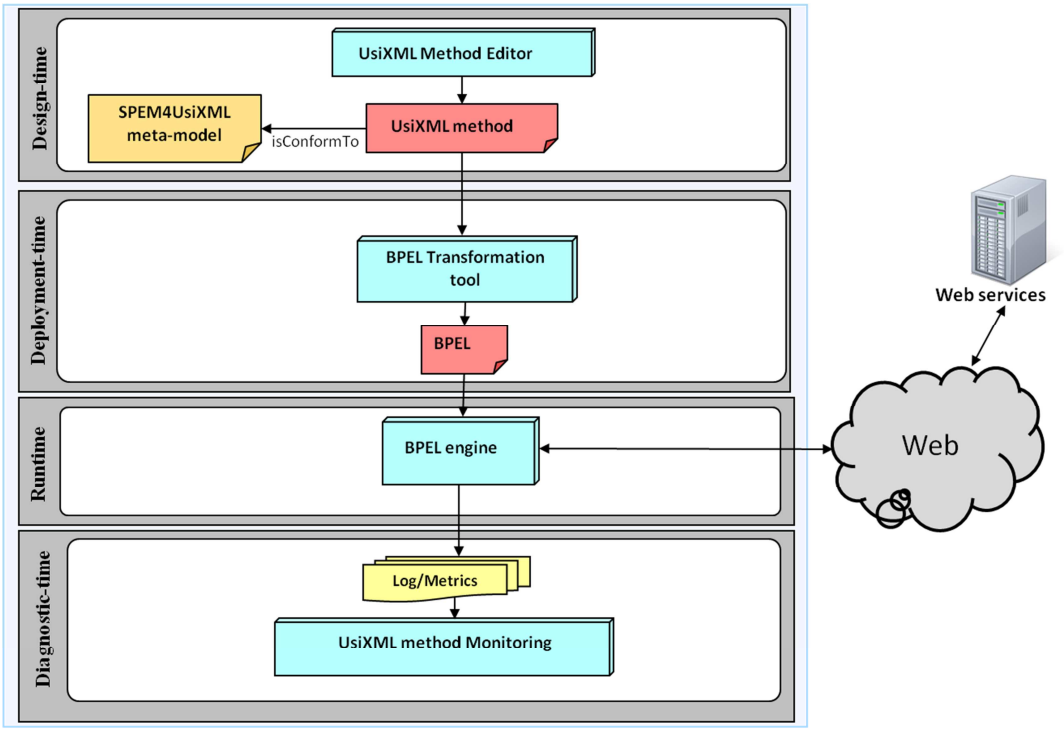

Figure 1. Overview of the software architecture for supporting the UsiXML methods

The proposed software architecture for supporting the UsiXML methods is made up of the following components:

1. **UsiXML Method editor:** allows the definition of a UsiXML method based on the SPEM4UsiXML meta-model.
2. **BPEL transformation tool:** allows the transformation of a SPEM4UsiXML definition into a BPEL process based on predefined transformation rules.
3. **BPEL engine:** allows the execution of the BPEL process by invoking the different transformation Web services.
4. **UsiXML method monitoring:** allows to control the enactment of SPEM4UsiXML methods based a historic model (log files). This historic model keeps trace of the enactment operations whenever they occur so that problems in a method can be identified based on predefined patterns (e.g. a delay in the execution of a step). Note that the description of this component and of the diagnostic-time is out of scope of this document.

In the next sections, we detail the different phases of the architecture.

1.2. Design-time

At design-time, the UsiXML method is defined by using the *UsiXML Method editor*. This definition needs to rely on a robust and well defined method meta-model in order to specify the elements of a UsiXML method. In the deliverable of the UsiXML project Task 2.1 [7], an analysis study has been conducted in order to compare the three method meta-model standards: OMG SPEM [5], OPEN [3] and ISO 247244 [6]. This study has concluded that these standard meta-models can be adopted to describe the UsiXML methods. However, it is more suitable to define a specific method meta-model in order to support the specific key elements of UsiXML methods (e.g. development path, development sub-step). For this reason, we have proposed in [7] a new meta-model for UsiXML methods. The proposed meta-model is based on SPEM 2.0. This choice is justified by the fact that SPEM 2.0 provides a great usability since it is a UML profile. Moreover, SPEM 2.0 contains generalization classes that allow the refinement of the vocabularies used to describe the concepts or the relationships between concepts. These abstract generalization classes allow creating a meta-model specific to the description of a UsiXML method. This specific meta-model is called SPEM4UsiXML. The SPEM4UsiXML extends the SPEM 2.0 ([5]) by adding new classes that allow describing the elements of UsiXML methods (e.g. development path, development sub-step).

The goal of the proposed meta-model, SPEM4UsiXML (SPEM for UsiXML), is to define the elements necessary for the description of any UsiXML method. In addition, like SPEM, SPEM4UsiXML separates the static aspect of a UsiXML method from the dynamic aspect of a UsiXML method. This means that SPEM4UsiXML reuses the UML diagrams (e.g. Class diagram, Activity diagram) for the definition of various UsiXML method concepts.

In this document we focus only on the description and the enactment of the dynamic aspect of the method. For this reason, we present in the following, the process structure package that represents the dynamic aspect of the SPEM4UsiXML meta-model. Note that a complete description of the SPEM4UsiXML meta-model is provided in the deliverable of the UsiXML project Task 2.1 [7].

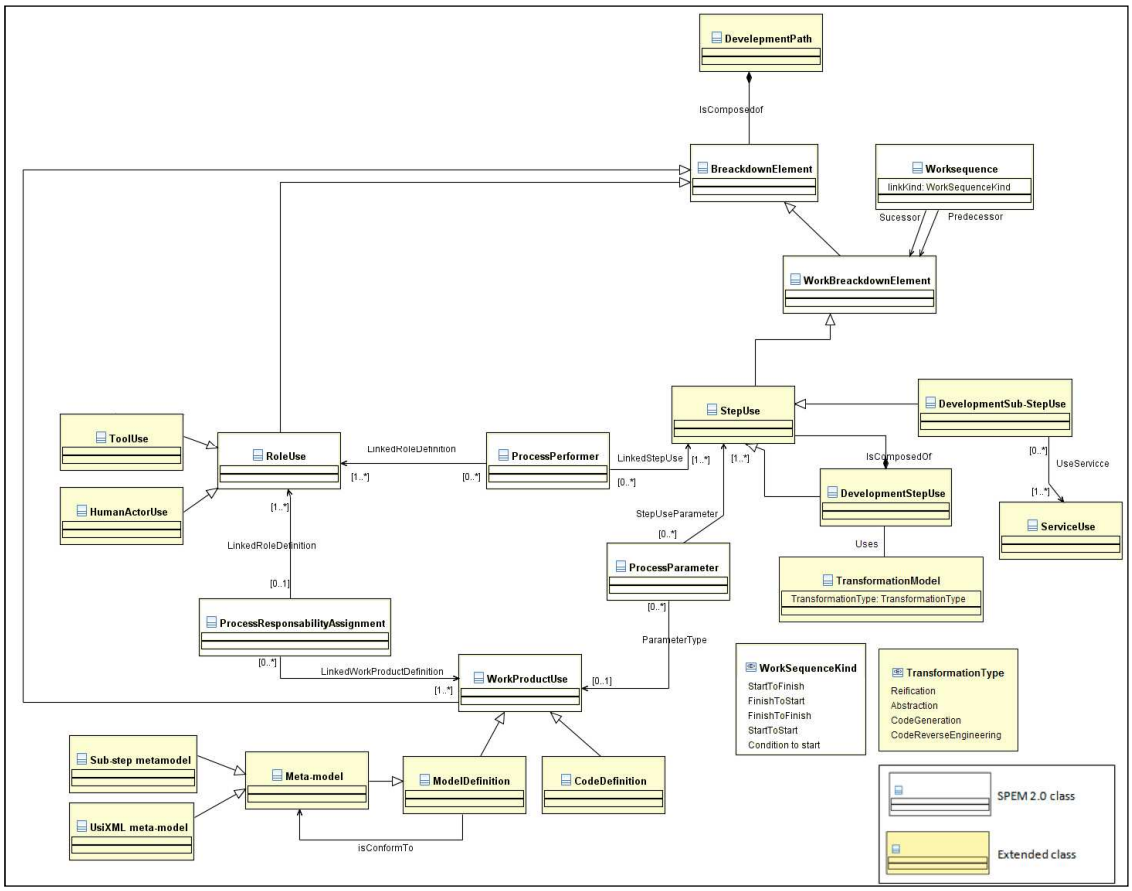

Figure 2. SPEM4UsiXML Process Structure package

As shown in Figure 2, SPEM4UsiXML adds new classes to the original SPEM process structure package in order to specify the control flow of the development steps and sub-steps and also the different products and producers used in the method process. The important classes of the SPEM4UsiXML process structure package are: *Development Path* defines the properties of a UsiXML method; *Development Step Use* defines the transformation steps of the UsiXML method that are performed by Role Use instances; *Development Sub-Step Use* that defines the sub-steps of a Development Step Use which can be achieved using a transformation Web service (Service Use), so that the enactment of the development sub-step is independent of any transformation system; *Role Use* represents a performer of a Development Step Use or a Development Sub-Step Use; and *Work Product Use* represents an input and/or output type for a Development Step (e.g. a model or UI code). The SPEM4UsiXML Method process structure package contains also some useful elements inherited from SPEM 2.0 like the *Work Sequence* class that represents a relationship between the different development (sub)steps in which one development (sub)step depends on the start or finish of another development (sub)step in order to begin or end.

Figure 3 gives an example of the UsiXML forward engineering method [4] expressed in SPEM4UsiXML. The starting point of this method is UsiXML task and domain models. These two models are then transformed into an abstract UI model which is then transformed into a concrete UI model. The concrete UI model is then used to generate UI code. For this reason, the UsiXML forward engineering method is composed of three development steps (two reifications and one

code generation). Theese development steps are represented, in Figure 3, by rectangles. Each development step is composed by a set of development sub-steps. Development sub-steps are represented by pentagons (e.g., identification of abstract UI structure, etc.). The development steps (and the development sub-steps) can be assigned to a producer who has a responsibility to execute or control the execution of the different development (sub)steps.

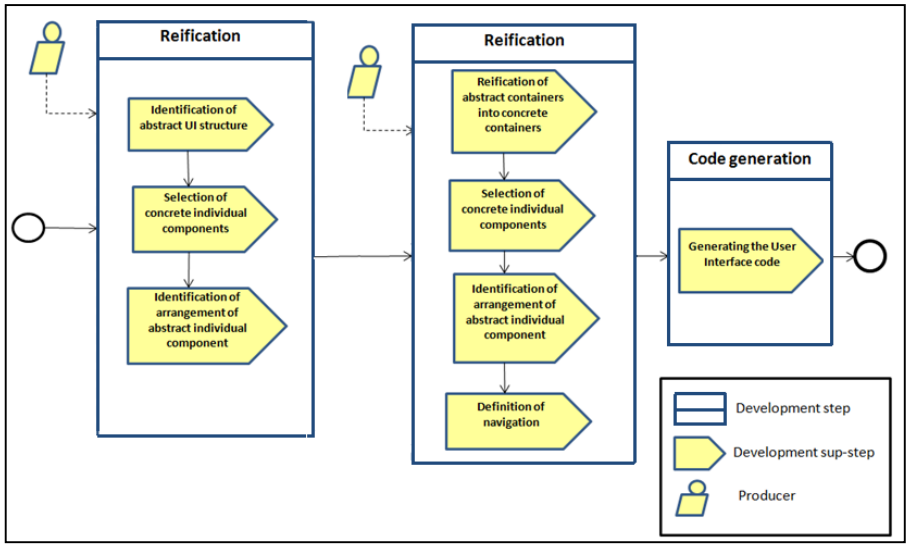

Figure 3. UsiXML Forward Engineering method expressed in SPEM4UsiXML

This UsiXML method needs to be enacted by a tool in order to support the transformation of the UsiXML models according to the method specification. However, the SPEM4UsiXML method meta-model provides a high level description, which is not precise enough to allow the execution of the UsiXML transformation. For this reason, in the deployment-time, the SPEM4UsiXML process needs to be mapped to a Web service composition execution language (called BPEL), as we will explain in the next section.

1.3. Deployment-time

The SPEM4UsiXML process package allows the description of a method process structure, but it does not introduce the formalism for enacting a method process. It rather proposes to reuse an existing externally-defined enactment model such as BPEL. Indeed, the separation of SPEM4UsiXML (like SPEM) method process structure from the behavior of the method process gives a method designer options to choose process behavior models that fits his/her needs. Although, the separation provides a flexible way to represent the behavioral aspects of SPEM processes, it does not define the mapping rules to link the elements of SPEM processes with the behavioral models. In the literature, several initiatives have been conducted to define mapping rules that allow automatically generating a specific executable model from a SPEM process (see [2] and [11]). For example, Feng et al. [11] propose a set of well-defined mapping rules to transform a SPEM process to a workflow expressed in XPDL [10]. Another example is the work proposed by Bendraou et al. in [2], which introduces transformation rules into BPEL.

According to the UsiXML FPP [9], the transformation engine will be implemented as a set of services. Each service enacts a specific development sub-step by using the associated

transformation rules. In this way, a UsiXML method can be seen as a Web services composition that can be enacted by using a BPEL engine. For this reason, a set of mapping rules should be defined in order to transform, at deployment-time, the elements of SPEM4UsiXML processes into elements of the OASIS standard BPEL [1]. In light of this, Table 1 proposes a set of mapping rules that map a subset of SPEM4UsiXML concepts with concepts of the BPEL language. These mappings rules are used by the BPEL transformation tool to transform SPEM4UsiXML specifications into BPEL processes.

Table 1: Mapping from SMEP4USiXML to BPEL

	SPEM4USiXML	BPEL	Description
Concept	Development Path	Process	A development path in SPEM4USiXML can be mapped to process in BPEL.
	Development Step	Scope Activity	Development step is a block which is composed of one or more development sub-steps. It can be mapped to scope activity in BPEL.
	Development Sub-step	Invoke Activity	A development sub-step is a concrete step where a service(s) is invoked (hence, it can be mapped to invoke activity in BPEL).
	Role	Partner Link	A role is an actor who executes an action(s). A role could be mapped to a Partner Link in BPEL.
	Product	Variable	Products of SPEM4USiXML are models and source code which can be represented using variables in BPEL.
Relationship	Start to Start	Flow Activity with Links	In order to start development step A, development step B must start first. This relationship can be expressed using flow and links.
	Start to Finish	Flow Activity with Links	Development step A needs to start before development step B finishes its activity. This relation could also be expressed using flow and links.
	Finish to Start	Sequence Activity	A sequence represents the sequences of execution of development sub-steps. It can be mapped to a sequence activity in BPEL.
	Finish to Finish	Flow Activity with Links	This relationship can also be expressed using flow and links to specify that development step A needs to be finished so as to B finish its activity.

	Condition to Start	If Activity	Only the subsequent activities whose condition evaluates to true are started. This relationship can be expressed as an BPEL If Activity.

Figure 4 illustrates the generated BPEL process for the UsiXML forward engineering method that was explained above.

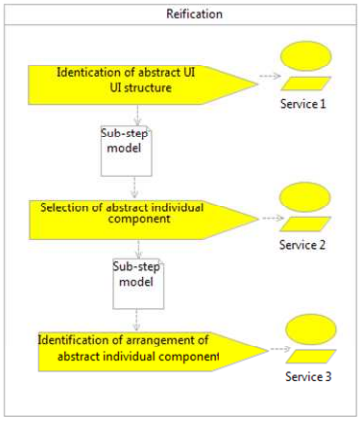

Figure 4. A. Part of the UsiXML forward engineering method expressed in SPEM4UsiXML

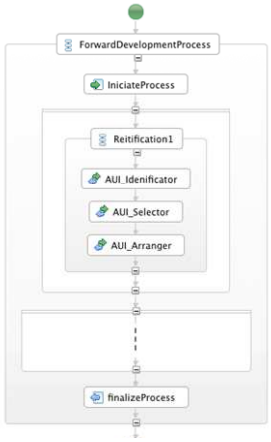

Figure 4. B. Part of the BPEL process of the UsiXML forward engineering method

Figure 4. Example of the transformation from SPEM4UsiXML to BPEL

In the next section, we explain how the generated BPEL process is executed by using a BPEL engine.

1.4. Runtime

At runtime, a BPEL engine interprets the fulfillment specification of the BPEL process of the UsiXML method. This interpretation is performed by invoking the different services that implement the UsiXML model transformation rules. The document [8] proposes a catalog of these transformation services. The proposed catalog considers two transformation services groups:

- *Vertical services* transform UsiXML models vertically according to the abstraction levels of the Cameleon Reference Framework, starting from the most abstract level to the most concrete one (Top-Down) or vice-versa (Bottom-up). An example of such services is the *TransformTDtoAUI* service (Top-Down) that transforms a task model and a domain model into an Abstract User Interface (AUI) model. Another example is the *TransformCUItoAUI* service (Bottom-up) that transforms a Concrete User Interface model (CUI) into an AUI model.
- *Horizontal services* transform UsiXML models horizontally according to the abstraction levels of the Cameleon Reference Framework, based on the user context. An example of

such services is the *TransformCAUItoAUI* service that implements the transformation from AUI model into another AUI model based on a Context model.

5. PROTOTYPE OF THE USIXML METHOD SUPPORT TOOL

To validate our proposed software architecture for supporting the UsiXML methods, we have developed a support tool that is dedicated to define and enact a UsiXML method. The tool is developed as an Eclipse plug-in that includes a SPEM4UsiXML model editor as well as a SPEM4UsiXML-to-BEPEL transformer engine. The speciation of this tool was published in the UIDL 2011 UsiXML workshop (See Annex A). Note that, the current version of the tool allows generating an abstract BPEL process without specifying the concrete transformation services. This is motivated by the fact that the UsiXML model transformation services are not implemented yet.

6. CONCLUSION AND FUTURE WORK

In this document, we proposed a software architecture for the definition and the enactment of the UsiXML methods. The architecture is based on the meta-model that is proposed in the Task 2.1 [7] for UsiXML method description (SPEM4UsiXML). This meta-model is based on the OMG standard, SPEM 2.0, which uses a UML profile to define elements of a method. The core elements of the SPEM4UsiXML are the development steps that are instances of transformation types. Development steps are decomposed into development sub-steps. A development sub-step can be executed by using a service. SPEM4UsiXML separates the operational aspect of a method (Method Content), from the temporal aspect of a methodology (Process Structure). This allows using any modeling language to describe the process behavior, like BPEL. Unfortunately, the SPEM4UsiXML meta-model cannot support the enactment of a UsiXML method on a specific endeavor. To deal with this limit, the proposed architecture transforms a SPEM4UsiXML model to a BPEL process so that a UsiXML method is considered as a Web service composition where each Web service enacts a specific development sub-step of the method. Consequently, a BPEL engine can be used to execute the SPEM4UsiXML models. However, BPEL language expresses a UsiXML method process in a fully automated way meaning that a human producer is not able to interact with the development sub-steps until the end of the process execution. For example, a human producer is not able to monitor the input to a development sub-step at runtime, s/he cannot cancel the process execution or s/he is not able to execute a development sub-step. For this reason, in the future work, we plan to address this problem by extending BPEL with a set of human interactions points in order to allow a human producer to interact with the method execution. This extension should allow the generation of a user interface for the UsiXML method in order to help the human producer to interact with the method at runtime. In addition, in the future, we also plan to develop a monitoring tool that allows to control the enactment of the SPEM4UsiXML methods based on a historic model. This historic model keeps trace of enactment operations whenever they occur so that problems in a method can be identified and corrected based on predefined patterns (e.g. a delay in the execution of a step).

7. REFERENCE

[1] Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guнzar, A., Kartha, N., Liu, C.K., Khalaf, R., Koenig, D., Marin, M., Mehta, V., Thatte, S., Rijn, D., Yendluri, P., Yiu, A.: Web services business

process execution language version 2.0 (OASIS standard). WS-BPEL TC OASIS, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html (2007)

[2] Bendraou, R., Combemale, B., Crégut, X., Gervais, M.P.: Definition of an executable SPEM 2.0. IEEE Computer Society (2007)

[3] Consortium, O.: OEPN (2010), http://www.open.org.au/

[4] Limbourg, Q., Vanderdonckt, J.: Multipath transformational development of user interfaces with graph transformations. In: Seffah, A., Vanderdonckt, J., Desmarais, M.C. (eds.) Human-Centered Software Engineering, pp. 107–138. Human-Computer Interaction Series, Springer London (2009), http://dx.doi.org/10.1007/978-1-84800-907-3\s\do5(6), 10.1007/978-1-84800-907-3\s\do5(6)

[5] OMG: Software Systems Process Engineering Meta-Model Specification version 2.0 (2008), In OMG Document Number: formal/08-04-02. Standard document URL: http://www.omg.org/spec/SPEM/2.0/PDF

[6] International Organization for Standardization / International Electrotechnical Commission, 2007. "ISO/IEC 24744. Software Engineering - Metamodel for Development Methodologies", JTC 1/SC 7, 2007

[7] UsiXML Project, 2011, "UsiXML methodology specification". In the deliverable of the Work package 2: Task 2.1, version 1.1, September 2011

[8] UsiXML Project, 2009, "Definition of a catalog of UsiXML services". In the deliverable of the Work package 2: Task 2.5, Version 0.1, August 2009

[9] UsiXML Project, 2010, UsiXML Full Project Proposal, Version 2.0, March 2010

[10] WFMC: Workflow management coalition workflow standard: Workflow process definition interface – XML process definition language (XPDL) (WFMC-TC-1025). Tech. rep., Workflow Management Coalition, Lighthouse Point, Florida, USA (2002)

[11] Yuan, F., Li, M., Wan, Z.: SEM2XPDL: Towards SPEM model enactment. In: Arabnia, H.R., Reza, H. (eds.) Software Engineering Research and Practice. pp. 240–245. CSREA Press (2006)

APPENDIX A

SUPPORT TOOL FOR THE DEFINITION & ENACTMENT OF THE USIXML METHODS

Mohamed Boukhebouze, Waldemar P. Ferreira Neto, Amanuel Koshima, Philippe Thiran, Vincent Englebert
PReCISE research center, University of Namur, Belgium,
{mboukheb, waldemar.neto, amanuel. koshima, philippe.thiran, vincent.englebert}@fundp.ac.be

1. ABSTRACT

In this paper, we propose a supporting tool for UsiXML methods based on a new meta-model called SPEM4UsiXML. This meta-model relies on the OMG standard SPEM 2.0 meta-model, which uses a UML profile to define the elements of a method. SPEM4UsiXML allows to express the core elements of the UsiXML methods (like development path, development step, and development sub-step). In addition, the meta-model separates the operational aspect of a UsiXML method (Method Content), from the temporal aspect of a method (Process Structure). Like SPEM, there is a lack of method enactments supporting in SPEM4UsiXML. To deal with this limitation, the proposed tool allows the enactment of the UsiXML methods by transforming a SPEM4UsiXML model to a BPEL model so that the a BPEL engine can be used to execute the transformed SPEM4UsiXML models.

1.1. KEYWORDS

UsiXML, SPEM, Method enactment, BPEL

2. INTRODUCTION

UsiXML (USer Interface eXtensible Markup Language) is a User Interface Description Languages (UIDL) that describes the user interface (UI) independently of any computing platform [13]. This independency is achieved by relying on the CAMELEON framework, which describes the UI at four main levels of abstractions: task & domain, abstract UI, concrete UI, and final UI. In UsiXML, the CAMELEON framework is realized by adopting a Model Driven Engineering (MDE) approach to specify a set of models representing the UI at different levels of abstraction. Besides, UsiXML uses a sets of transformations to derive a UI model from another model. For example, a high-level model (e.g. task & domain model) can be transformed into low-level analysis or design model (e.g. concrete UI model) [8]. Another example of a UsiXML transformation is the extraction of high-level model from a set of low-level models or from code [8].

According to Limbourg et al. in [8], UsiXML transformations may be combined to form a UsiXML method. A UsiXML method, which is also called development path [8], is the process to follow for developing a user interface based on UsiXML models. In a UsiXML method, transformations are considered as development steps that can be decomposed into nested development sub-steps. In turn, a development sub-step realizes a basic goal assumed by a developer while constructing a UI.

To reap all the benefits, a UsiXML method needs to be designed and evaluated by describing formally its content (its semantics) and its form (its abstract/concrete syntax). For this reason, a UsiXML method needs to be compliant with a well-defined meta-model so that the core elements of UsiXML methods (e.g. development path, development step and development sub-step) can be formally defined. In addition, the enactment of UsiXML methods needs to be supported by a tool. By enactment of a UsiXML method we mean the ability of a tool to support the UsiXML models transformation according to the method specification.

In order to achieve the UsiXML method enactment with a tool, the UsiXML method meta-model needs to be expressiveness to allow the execution of the UsiXML transformation.

In this paper, we propose a support tool method that allows the definition and the enactment of UsiXML methods. The definition of a UsiXML method (in this tool) is based on a SPEM meta-model [10]. SPEM is an OMG standard that provides a great usability using UML profiles. In addition, it contains generalization classes that allow the refinement of the vocabularies used to describe the concepts or the relationships between concepts. In order to support the specific key elements of the UsiXML methods (e.g. development path, development step and development sub-step), our proposed tool uses a SPEM meta-model specific to UsiXML methods. This specific meta-model is called SPEM4UsiXML.

Like SPEM, the SPEM4UsiXML meta-model allows the description of a method process structure without introducing its own formalism to precisely describe the process behavior models. [10] argues that the separation of SPEM method structure from the behavior of the method opens up the possibility to reuse existing externally-defined behavior models. A method described with the SPEM 2.0 meta-model can be enacted by mapping it to a business flow or an execution language such as BPEL [1] or XPDL [14] and then executing this representation of processes using enactment engine such as a BPEL engine [10]. In order to provide a flexible and independent transformation systems, this work implements UsiXML model transformation engine as Web services. Each Web service enacts a specific development sub-step by using associated transformation rules. In this way, a UsiXML method can be seen as a Web services composition. Our method support tool allows the enactment of a UsiXML method by transforming a SPEM4UsiXML model to a BPEL based on a set of mapping rules and by executing it using a BPEL engine.

The rest of the paper is organized as follows. Section 2 gives an overview of UsiXML methods. Section 3 introduces the SPEM4UsiXML meta-model. Section 4 presents the transformation of a SPEM4UsiXML model to a BPEL. Section 5 demonstrates the prototype of the support tool for the UsiXML methods. Finally, the paper end with a conclusion and future works.

3. USIXML METHODS

In this section, the background definition for UsiXML methods is given. A UsiXML method is a process that transforms progressively the UsiXML models in order to obtain specifications that are detailed and precise enough to be rendered or transformed into code [8]. A UsiXML method is also used to synthesize abstract models from detailed models. To achieve the UsiXML transformations, different types of transformation mechanisms can be used [8]:

- Reification is a transformation of a high-level model into a low-level model.
- Abstraction is a transformation that extracts a high level model from a set of low-level models.
- Translation is a same level models transformation based on a context of use change. In this work, the context of use is defined as a triple of the form (E, P,U) where E is a possible or actual environments considered for a software system, P is a target platform, U is a user category.
- Code generation is a process of transforming a concrete UI model into a source code.
- Code reverse engineering is the inverse process of code generation.

These different UsiXML transformation types are instantiated by development steps [8]. These development steps may be combined to form a UsiXML method. The process of combining development steps into a UsiXML method is called a development path. Vanderdonckt et al. identifies several types of development paths, for example [8]:

- Forward engineering is a composition of reification(s) and code generation enabling a transformation of a high-level viewpoint into a lower level viewpoint.
- Reverse engineering is a composition of abstraction(s) and code reverse engineering, which enables a transformation of a low-level viewpoint into a higher-level viewpoint.
- Context of use adaptation is a composition of a translation with another type of transformation that enables a viewpoint to be adapted in order to reflect a change of context of use of a UI.

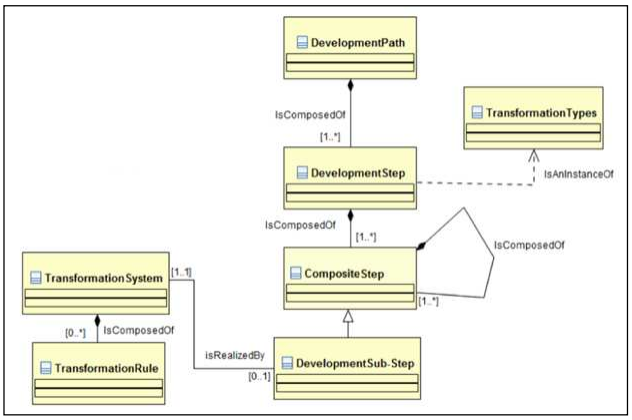

Figure A.1. Transformation path, step and sub- step [8]

Figure A.1 represents an overview of the UsiXML method meta-model. This meta-model assumes that the development steps are decomposed into nested development sub-steps. A development sub-step may consist of activities to select concrete interaction objects, navigation objects, etc. This could be realized by a transformation mechanism (e.g. graph transformation [11] and [3]) based on sets of transformation rules [11]. Composite Step is a generalization class that is used to express a development path in a tree-structure. It represents a set of development sub-steps as leafs and a development step as root of a tree.

Based on the meta-model shown in *Figure A.1*, three major elements of the UsiXML method are considered such as work, product and producer.

- The work represents what must be done. It is defined in terms of development step and development sub step
- The product represents the artifact that must be manipulated by a development step and a development sub step (i.e. created, used or changed). It can concern a UI model or a UI code. In turn, a model can be a UsiXML model that is used/generated by a development step or a sub-step model that is used/generated by a development sub-step.
- The producer represents the agent that has the responsibility to execute a work unit. It is defined in terms of person, role, team, tool, service, etc.

Figure A.2 shows an illustration of the forward engineering method. This method is fully explained in [11]. The starting point of the forward engineering is a task and a domain model (products). These models are transformed into an abstract UI based on the transformation rules specified in works. Afterwards, the abstract UI model is transformed into a concrete UI model (products). Finally, the code is generated (products). In order to achieve these transformations, a sequence of development steps (sequence of reification and code generation) needs to be performed. Each development step may involve a set of development sub-steps. For example, the first development step involves a development sub-step like identification of Abstract UI structure. This sub-step consists in the definition of groups of abstract interaction objects (an element of the abstract user interface). Each group of abstract interaction objects corresponds to a group of tasks (in task model), which are tightly coupled together. To achieve its work, the sub-step can use a sequence of rules. For example, identification of Abstract UI structure uses sequences of two rules; R1: for each leaf task of a task tree, create an Abstract Individual Elements; and R2: create an Abstract Container structure similar to the task decomposition structure. Indeed, each development step takes a UsiXML model(s) as input and transform it to another UsiXML model(s) by involving a set of development sub-steps, which in turn manipulates sub-steps models by using a set of rules. Note that, each development step (and development sub-step) has a producer responsible of their execution. For example, the first development step can have a human actor who verifies the transformation done in this step. Whereas a transformation tool can execute the rules sequence of the sub-step "*identification of abstract UI structure*".

In the next section, we present our proposed meta-model for the UsiXML method, SPEM4UsiXML.

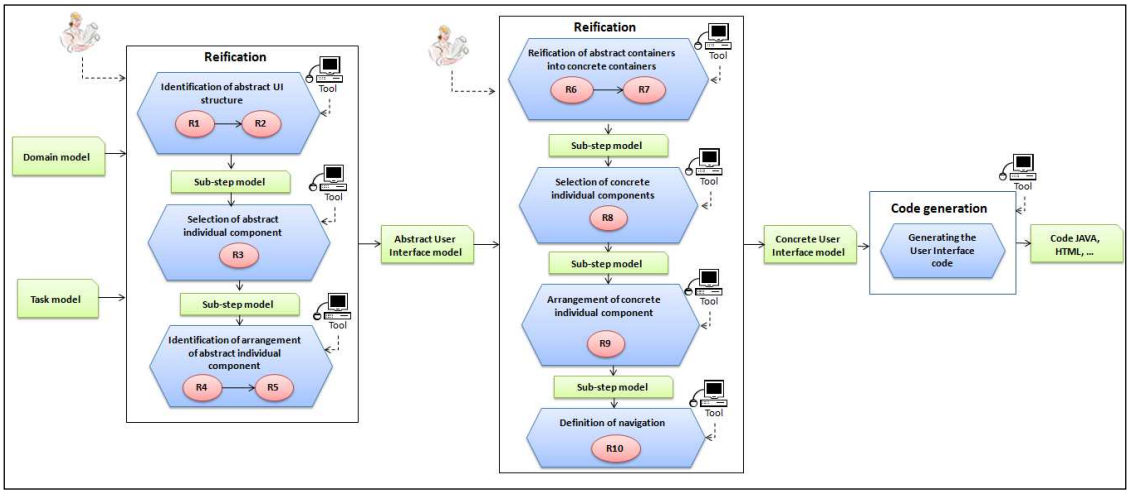

Figure A.2. Forward Transformational Development of UIs

4. SPEM4USIXML

UsiXML User interface designers need to rely on robust and well defined method meta-model in order to specify the elements of a UsiXML method. In the literature, several method standard meta-models have been introduced like SPEM [10], OPEN [4] and ISO 24744 [12]. These standards describe the core elements of a method in different ways. Each standard is built on different main principles. SPEM 2.0 [OMG 2008] is an OMG standard that reuses the UML diagrams to describe the elements of a method. Whereas OPEN [OPF 2005] defines an industry-standard meta-model that provides a significant detail to describe different elements of a method. However, both SPEM and OPEN standards do not support the method enactment. ISO 72444 [12] uses a dual-layer modelling to allow the method engineer to configure the enactment of the method from the meta-model level by using the Clabject and the Powerptype concepts. However, the object-oriented programming languages (like JAVA) do not support the dual-layer ([5] and [7]).

Although these standard meta-models can be adopted to describe the UsiXML methods, it is more suitable to define a specific method meta-model in order to support the specific key elements of the UsiXML methods (e.g. development path, development sub-path). For this reason, we propose in this paper a new meta-model for the UsiXML methods.

The proposed meta-model is based on SPEM 2.0. This choice is justified by the fact that SPEM 2.0 provides a great usability since it is a UML profile. Moreover, SPEM 2.0 contains generalization classes that allow the refinement of the vocabularies used to describe the concepts or the relationships between concepts. These abstract generalization classes allow creating a UsiXML method meta-models specific to a certain domain (e.g. User Interface Development)

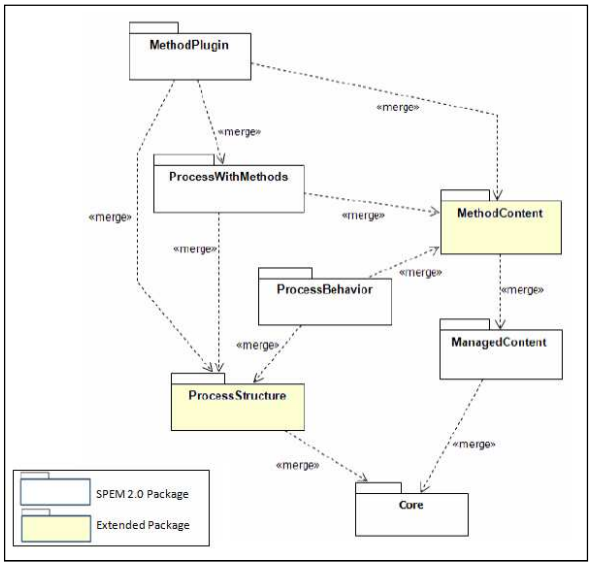

Figure A.3. Structure of the SPEM4UsiXML meta-model

The goal of the proposed meta-model, SPEM4UsiXML (SPEM for UsiXML), is to define the elements necessary for the description of any UsiXML method. The SPEM4UsiXML extends the SPEM 2.0 ([10]) by adding new classes. In addition, like SPEM, SPEM4UsiXML separates the operational aspect of a UsiXML method from the temporal aspect of a UsiXML method. This means that SPEM4UsiXML reuses the UML diagrams for the presentation of various UsiXML method concepts. As depicted in *Figure A.3,* the SPEM4UsiXML meta-model uses seven main meta-model packages inherited from SPEM: *Method Content* describes the operations aspect of a UsiXML method; *Process Structure and Process Behaviour* describes the temporal aspect of a UsiXML method, *Process With Methods* describes the link between these two aspects; Core provides the common classes that are used in the different packages; *Method Plug-in* describes the configuration of a UsiXML method; *Managed Content* describes the documentation of a UsiXML method.

SPEM4UsiXML extends the classes of the *Method Content* and the *Process Structures*. Indeed, SPEM4UsiXML adds new classes for the SPEM method content meta-model package in order to specify several development steps, sub sub-steps, products and producers. Moreover, SPEM4UsiXML adds new classes in the SPEM process structure package in order to specify the control flow of development steps, sub-steps, products and producers that are used in the UsiXML method process.

In this paper we focus only on the description and the enactment of the dynamic aspect of the method (i.e. method process). For this reason, we present, the *Process Structure* package of SPEM4UsiXML in the next section.

4.1. Process Structure Package

As shown in *Figure A.4*, SPEM4UsiXML adds new classes to the SPEM *Process Structure* package. The white classes represent the classes of SPEM that are not modified, whereas the yellow classes represent the classes extended by SPEM4UsiXML.

- *Breakdown Element*: is a generalization class that defines a set of properties used by the element of a UsiXML method (Product, Development step and producer).
- *Development Path*: defines the properties of a UsiXML method.
- *Work Breakdown Element*: provides specific properties for Breakdown Elements that represent a *Development Step* and a *Development Sub-Step*.
- *Step Use*: is a generalization class that defines a set of properties used by the element of the Development Step, the Composite Step and the Development Sub-Step.
- *Composite Step Use*: is a generalization class that is used to define a tree-structure with a set of development sub-step as a leaf and a development step as the root.
- *Development Step Use*: defines the transformation steps of the UsiXML method that are performed by Roles Use instances. A Development Step Use is associated to an input and an output Work Products Use.
- *Development Sub-Step Use*: defines the sub-steps of a Development Step Use. As sub-step can be achieved using a autonomous component called service (Service Use), so that the enactment of the development sub-step is independent of any transformation system.
- *Role Use*: represents a performer of a Development Step Use or a Development Sub-Step.
- *Work Product Use*: represents an input and/or output type for a Development Step. It can concern a model (Model Use) or a code (Code Use).

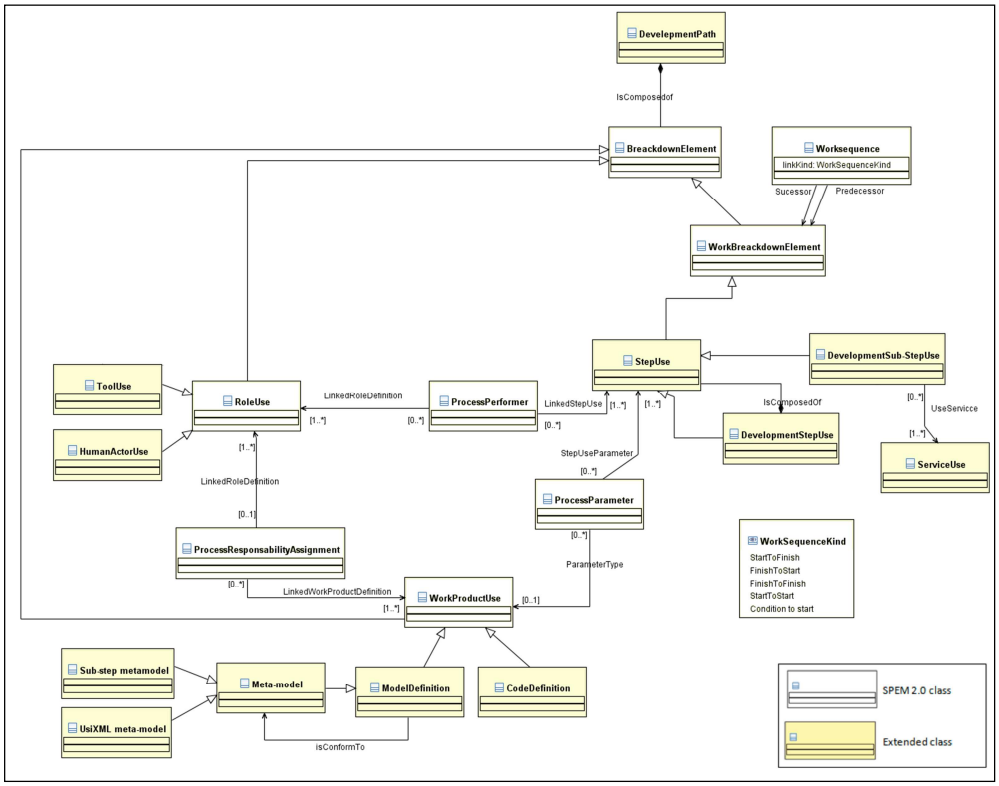

Figure A.4. SPEM4UsiXML Process Structure package

The SPEM4UsiXML Method process structure package contains also some useful elements inherited from SPEM 2.0 like:

- *Process Responsibility Assignment:* links Role Uses to Work Product Uses by indicating that the Role Use has a responsibility relationship with the Work Product Use.
- *Process Performer:* links Role Uses to Development Step Use by indicating that these Role Use instances participate in the work defined by the Development Step Use.
- *Work Sequence:* represents a relationship between two Work Breakdown Elements in which one Work Breakdown Elements depends on the start or finish of another Work Breakdown Elements in order to begin or end. Indeed, a Work Sequence has 4 kinds:
  - *StartToStart* expresses that a Work Breakdown Element (B) cannot start until a Work Breakdown Element (A) start;
  - *StartToFinish* expresses that a Breakdown Element (B) cannot finish until a Work Breakdown Element (A) starts;
  - *FinishToStart* expresses that a Work Breakdown Element (B) cannot start until a Work Breakdown Element (A) finishes;
  - *FinishToFinish* expresses that a Work Breakdown Element (B) cannot finish until a Work Breakdown Element (A) finishes.
  - *ConditionToStart* expresses that a Work Breakdown Element can be started only if the condition is satisfied.

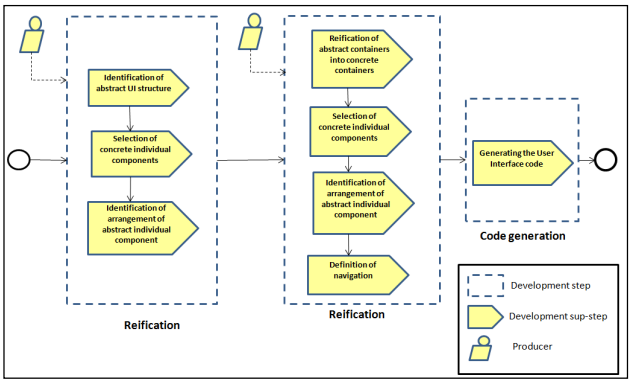

Figure A.5. UsiXML Forward Engineering method expressed in SPEM4UsiXML

Figure A.5 gives an example of a forward engineering method expressed in SPEM4UsiXML. In this method, various development steps are represented by dashed rectangles. Each development step can be composed by a set of development sub-steps. Development sub-steps are represented by pentagon (e.g. identification of an abstract UI structure, etc.) The development steps (and the development sub-steps) can be assigned to a producer who has a responsibility to execute or control an execution of the different development (sub)steps.

This UsiXML method needs to be enacted by a tool in order to allow supporting the transformation of the UsiXML models according to the method specification. However, the SPEM4UsiXML method meta-model provides a high level description, which is not precise enough to allow the execution of the UsiXML transformation. For this reason, the SPEM4UsiXML process needs to be mapped to an execution language. In the next section, we detail the mapping of SPEM4UsiXML process to a BPEL process.

5. USIXML METHOD ENACTMENT

SPEM4UsiXML process package allows the description of a method process structure, but it does not introduce the formalism for enacting a method process. It rather proposes to reuse an existing externally-defined an enactment model such as BPEL.

For this reason, in the next section, we detail how we can map SPEM4UsiXML process to a BPEL process. The separation of SPEM4UsiXML (like SPEM) method process structure from the behavior of the method process opens up the possibility to utilize enactment machines for many different kinds of behavior modeling approaches [10]. The motivation behind this separation is to give a method designer options to choose process behavior models that fits his/her needs. Although, the separation provides a flexible way to represent the behavioral aspects of SPEM processes, it does not define the mapping rules to link the elements of SPEM process with the behavioral models. In the literature, several initiatives have been conducted to define mapping rules that allow automatically generating a specific executable model from a SPEM process [15] and [2]. For example, Feng et al. [15] propose a set of well-defined mapping rules to transform a SPEM process to a workflow expressed in XPDL [14]. Another example is the work proposed by Bendraou et al. in [2], which introduces transformation rules into BPEL.

Because SPEM4UsiXML extends SPEM with additional classes that specify elements of a UsiXML method (e.g. development steps and sub sub-steps), a set of mapping rules should be defined in order to link the elements of SPEM4UsiXML process with the OASIS standard BPEL. Indeed, a UsiXML process can be considered as a Web service composition orchestration where each Web service enacts a specific development sub-step transformation so that the transformation will be flexible and independent to any transformation system. As a result, an enactment machine for BPEL models can be used to run a UsiXML method. In light of this, we propose a set of mapping rules between a subset of SPEM4UsiXML concepts and the BPEL language in *Table 1*.

Table 1: Mapping from SMEP4USiXML to BPEL

	SPEM4USiXML	BPEL	Description
Concept	Development Path	Process	A development process in SPEM4USiXML can be mapped to process in BPEL.
	Development Sub	Scope Activity	Development step is a block which is composed of one or more development sub-steps. It can be mapped to Scope in BPEL.
	Development Sub-steps	Invoke Activity	A development sub-step is a concrete step where a service(s) is invoked, hence, it can be mapped to invoke acvtivity in BPEL.
	Role	Partner Links	A role is an actor who executes an action(s). A role could be mapped to a parent link in BPEL.
	Product	Product	Products of SPEM4USiXML are models and source codes which can be represented using variables in BPEL.
Relationship	Start to Start	Flow Activity with Links	In order to start development step A, development step B must start first. This relationship can be expressed using flows.
	Start to Finish	Flow Activity with Links	Development step A needs to start before development step B finishes its activity. This relation could also be expressed using flow and links.
	Finish to Start	Sequence Activity	A sequence represents the sequences of execution of development sub-steps. It can be mapped to a sequence in BPEL..
	Finish to Start	Flow Activity with Links	This relationship can also be expressed using flow and links to specify development step A needs to be finished so as to B finish its activity.
	Condition to Start	If Activity	Only the subsequent activities that the condition is true are started. This relationship can be expressed as an If Activity where the condition is the <If Expression>.

6. USIXML METHOD SUPPORT TOOL

This section describes the UsiXML support tool that is dedicated to define and enact a UsiXML method. The tool is developed as an Eclipse plug-in that includes a SPEM4UsiXML model editor as well as a SPEM4UsiXML-to-BEPEL transformer engine.

Figure A.6 shows a screenshot of the SPEM4UsiXML model editor that is build based on the Eclipse Graphical Modeling Framework (GMF) [9]. This framework provides a generative component and a runtime infrastructure for developing graphical editors based on a well-defined meta-model.

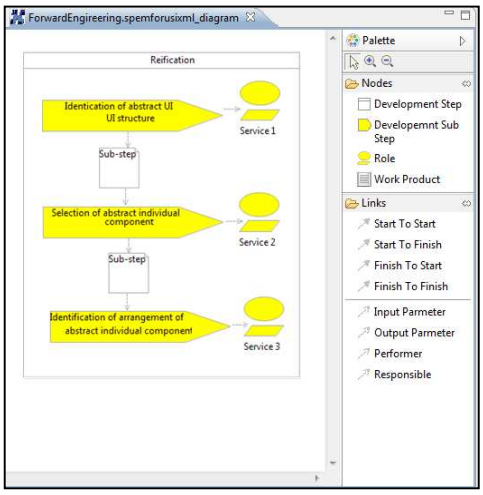

Figure A.6. *Screenshot of the SPEM4UsiXML model editor*

The UsiXML support tool is based on an ATL transformation language to specify the mapping between a SPEM4UsiXML method and a BPEL process. The mapping rules are described and executed using the ATL toolkit. The ATL toolkit [6] is a model transformation tool that allows to generate a target model from a source model based on mapping rules. *Figure A.7* illustrates the generated BPEL process for the UsiXML forward engineering method that was explained above.

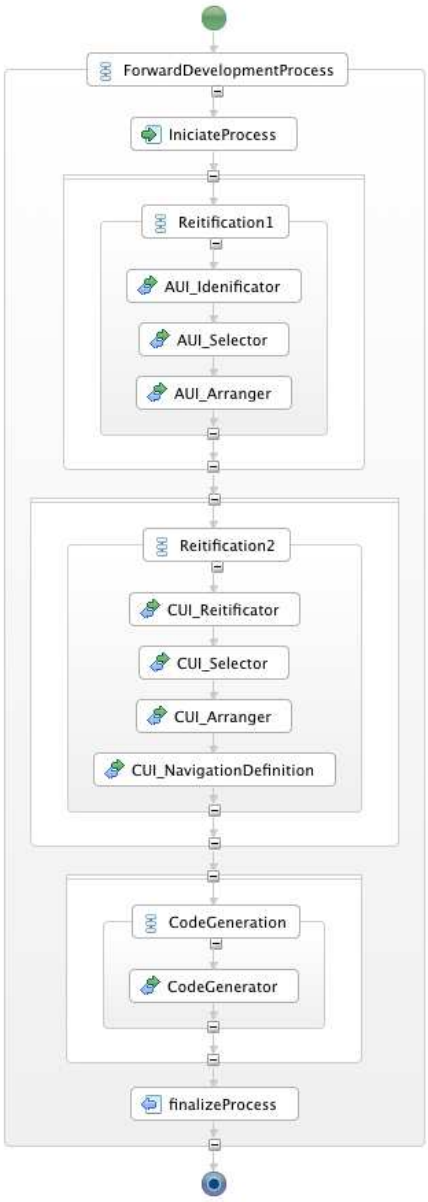

Figure A.7. *The BPEL Process of the UsiXML forward engineering method.*

7. CONCLUSION AND DISCUSSIONS

In this paper, we proposed a support tool for the definition and the enactment of the UsiXML methods. The tool is based on a new meta-model for UsiXML method description, called SPEM4UsiXML. This meta-model is based on the OMG standard, SPEM 2.0, which uses a UML profile to define elements of a method. The core element of the SPEM4UsiXML is the development steps that are instances of transformation types. Development steps are decomposed into development sub-steps. A development sub-step can be executed by using a Web service. SPEM4UsiXML separates the operational aspect of a method (Method Content), from the temporal aspect of a methodology (Process Structure). This allows using any modeling language to describe the process behavior like BPEL. Unfortunately, the SPEM4UsiXML meta-model cannot support the enactment of a UsiXML method on a specific endeavor. To deal with this limit, the proposed support tool (for UsiXML methods) transforms a SPEM4UsiXML model to a BPEL process so that a UsiXML method is considered as a Web service composition where each Web service enacts a specific development sub-step of the method. Consequently, a BPEL engine can be used to execute the SPEM4UsiXML models. However, BPEL language expresses a UsiXML method process in a fully automated way meaning that a human producer is not able to interact with the development sub-steps until the end of the process execution.

For example, a human producer is not able to monitor the input to a development sub-step at runtime, s/he cannot cancel the process execution or s/he is not able to execute a development sub-step. For this reason, in the future work, we plan to address this problem by extending BPEL with set of human interactions points in order to allow a human producer to interact with the method execution. This extension should allow the generation of a user interface for the UsiXML method in order to help the human producer to interact with the method at runtime. In addition, in the future, we also plan to develop a monitoring tool that allows to control the enactment of the SPEM4UsiXML methods based a historic model. This historic model keeps trace of enactment operations whenever they occur so that problems in a method can be identified and corrected based on predefined patterns (e.g. a delay in the execution of a step).

8. REFERENCES

[1] Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guнzar, A., Kartha, N., Liu, C.K., Khalaf, R., Koenig, D., Marin, M., Mehta, V., Thatte, S., Rijn, D., Yendluri, P., Yiu, A.: Web services business process execution language version 2.0 (OASIS standard). WS-BPEL TC OASIS, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html (2007)

[2] Bendraou, R., Combemale, B., Crégut, X., Gervais, M.P.: Definition of an executable SPEM 2.0. IEEE Computer Society (2007)

[3] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting with Computers 15(3), 289–308 (2003), http://dx.doi.org/10.1016/S0953-5438(03)00010-9

[4] Consortium, O.: OEPN (2010), http://www.open.org.au/

[5] Gutheil, M., Kennel, B., Atkinson, C.: A systematic approach to connectors in a multi-level modeling environment. In: Czarnecki, K., Ober, I., Bruel, J.M., Uhl, A., Völter, M. (eds.) MoDELS. Lecture Notes in Computer Science, vol. 5301, pp. 843–857. Springer (2008), http://dx.doi.org/10.1007/978-3-540-87875-9\s\do5(5)8

[6] Jouault, F., Kurtev, I.: Transforming models with atl. In: Bruel, J.M. (ed.) Satellite Events at the MoDELS 2005 Conference, Lecture Notes in Computer Science, vol. 3844, pp. 128–138. Springer Berlin / Heidelberg (2006), http://dx.doi.org/10.1007/11663430\s\do5(1)4, 10.1007/11663430\s\do5(1)4

[7] Kuehne, T., Schreiber, D.: Can programming be liberated from the two-level style: multi-level programming with deepjava. In: OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications. pp. 229–244. ACM, New York, NY, USA (2007), http://dx.doi.org/10.1145/1297027.1297044

[8] Limbourg, Q., Vanderdonckt, J.: Multipath transformational development of user interfaces with graph transformations. In: Seffah, A., Vanderdonckt, J., Desmarais, M.C. (eds.) Human-Centered Software Engineering, pp. 107–138. Human-Computer Interaction Series, Springer London (2009), http://dx.doi.org/10.1007/978-1-84800-907-3\s\do5(6), 10.1007/978-1-84800-907-3\s\do5(6)

[9] Moore, B., Organization, I.B.M.C.I.T.S., ebrary, I.: Eclipse development using the graphical editing framework and the eclipse modeling framework. IBM, International Technical Support Organization (2004)

[10] OMG: Software Systems Process Engineering Meta-Model Specification version 2.0 (2008), In OMG Document Number: formal/08-04-02. Standard document URL: http://www.omg.org/spec/SPEM/2.0/PDF

[11] Stanciulescu, A.: A Methodology for Developing Multimodal User Interfaces of Information System. Ph.D. thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium (June 2008)

[12] International Organization for Standardization / International Electrotechnical Commission, 2007. “ISO/IEC 24744. Software Engineering - Metamodel for Development Methodologies”, JTC 1/SC 7, 2007

[13] UCL: Usixml v1.8 reference manual (February 2007), iTEA2, UsiXML Full Project Proposal

[14] WFMC: Workflow management coalition workflow standard: Workflow process definition interface – XML process definition language (XPDL) (WFMC-TC-1025). Tech. rep., Workflow Management Coalition, Lighthouse Point, Florida, USA (2002)

[15] Yuan, F., Li, M., Wan, Z.: SEM2XPDL: Towards SPEM model enactment. In: Arabnia, H.R., Reza, H. (eds.) Software Engineering Research and Practice. pp. 240–245. CSREA Press (2006)