ITEA 2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

WORKPACKAGE 2: METHOD ENGINEERING

D2.1 V2

USIXML METHOD SPECIFICATION

Project acronym: UsiXML

Project full title: User interface eXtensible Mark-up Language

ITEA label n°08026

DOCUMENT CONTROL

***Deliverable N°**: D2.1*
***Due Date**: 10/2011*
***Delivery Date**: 10/2011*

***Short Description**:* The task 2.1 will bridge the gap between the UI modelling perspective and software development processes used by the technology providers. Relevant characteristics of these processes will be extended to address μ^7-compatible applications and merged into a unified software engineering method for interactive applications

***Lead Partner**: UND*
***Contributors:** UND, UJF, ICI, UPV*
***Made available to**:*

Rev	Date	Author	Checked by	Internal Approval	Description
0.1	15/09/10	Draft – Mohamed Boukhebouze, Philippe Thiran, UND.			Initial version
1.0	30/09/10	Final deliverable V1.0 - Mohamed Boukhebouze, Philippe Thiran, UND.	Vincent Englebert, UND.		Reviewed after the Meeting with Vincent Englebert
1.1	06/10/10	Draft – Mohamed Boukhebouze, Philippe Thiran, UND.			Minor revision – Eric Céret, Sophie Dupuy- Chessa, UJF
1.2	16/10/10	Draft – Mohamed Boukhebouze, Philippe Thiran, UND.			Reviewed after the Task 2.1 Audio Meeting
1.3	01/02/11	Draft – Mohamed Boukhebouze, Philippe Thiran, UND.			Reviewed after the general assembly of UsiXML

1.4	03/02/2011	Draft – Mohamed Boukhebouze, Philippe Thiran, UND.			Reviewed after the review of Costin Pribeanu (ICI)
1.5	04/06/2011	Draft – Mohamed Boukhebouze, Philippe Thiran, UND.			Reviewed after the review of Nathalie Aquino (UPV)
2.0	30/09/2011	Final - deliverable V2.0 Mohamed Boukhebouze, Philippe Thiran, UND.			New version of the UsiXML method meta-model (SPEM4UsiXML); New section introducing the enactment of the UsiXML methods is added

CONTENTS

1. EXECUTIVE SUMMARY

The task 2.1 of the UsiXML project aims to provide a unified software engineering method for User Interface based on UsiXML language. This method describes the process to follow during a User Interface design. The method needs to be compliant with a well-defined meta-model so that we formally define the core elements of the UsiXML method. In this document, we propose a SPEM based meta-model for UsiXML method description.

2. DOCUMENTS

2.1. Reference

D2.1 UsiXML	

3. INTRODUCTION

UsiXML (USer Interface eXtensible Markup Language) is a User Interface Description Language (UIDL) that uses Model-Driven Engineering (MDE) for specifying a User Interface (UI) at an implementation-independent level [1]. The UI specifications are usually specified in different models. Each UI level is described by a model(s). UsiXML is based on the Cameleon reference framework [2]. This framework describes a UI in 4 main levels of abstraction: task & domain level, abstract UI level, concrete UI and final UI (see Figure 1). On the basis of these 4 levels, UsiXML proposes a set of models (e.g. task model, domain model, abstract user interface model, etc.). Note that the complete set of the UsiXML models is provided in [3]. The MDE approach allows developing the UsiXML UI by transforming progressively the UsiXML models to obtain specifications that are detailed and precise enough to be rendered or transformed into code [4]. For this reason, the UsiXML development method is a transformation process based on the Cameleon reference framework. Figure 1 illustrates the different types of transformations in the Cameleon framework [5]:

- *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.

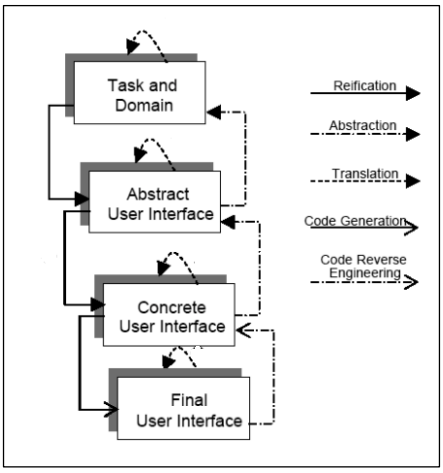

Figure 1. Different kinds of transformations steps of UsiXML [5]

According to Vanderdonckt et al. in [5], the different transformation types are instantiated by development steps. These development steps may be combined to form development paths. A development path is the process to follow for developing a user interface based on UsiXML. Several types of development paths are identified:

- *Forward engineering* (or requirement derivation) is a composition of reifications and code generation enabling a transformation of a high-level viewpoint into a lower level viewpoint.
- *Reverse engineering* is a composition of abstractions and code reverse engineering enabling 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 enabling a viewpoint to be adapted in order to reflect a change in the context of use of a UI.

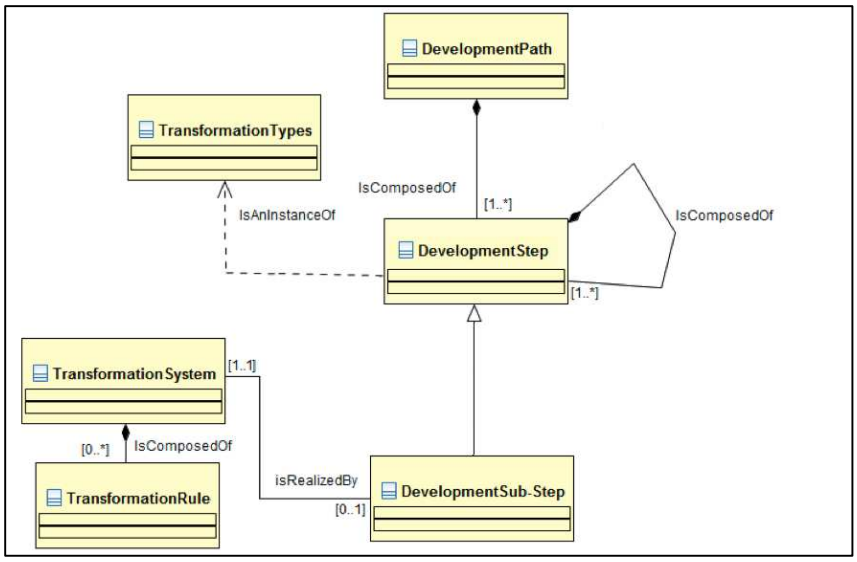

Figure 2. Transformation path, step and sub-step

Figure 2 represents an overview of the UsiXML method meta-model. This meta-model assumes that development paths are composed of development steps. In turn, development steps are instances of transformation types. Development steps are decomposed into nested development sub-steps. A development sub-step realizes a basic goal assumed by the developer while constructing a system. It may consist, for instance, to select concrete interaction objects, defining navigation, etc. Development sub-steps may be realized by a transformation system (e.g. graph transformation [4, 5]) based on transformation rules [5]. Note that, a development step can be composed of nested development steps. In another word, a development step can be represented as a tree-structure with a set of development sub-steps as leafs and a development step as root.

According to the several types of development paths, three major elements of the UsiXML based User Interface development method can be considered (see Figure 3):

- *The work* represents what must be done. It is defined in terms of development step and development sub step.
- *The product* represents the artefact that must be manipulated by development step and development sub step (i.e. created, used or changed). It can concern models and 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, etc.

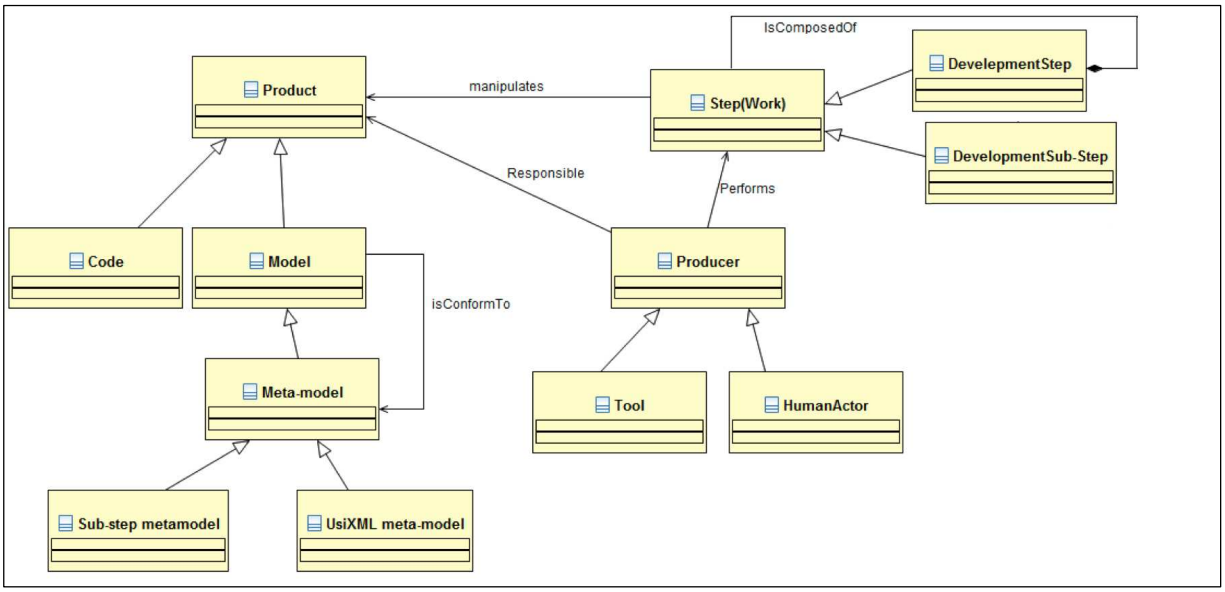

Figure 3. The Core elements of the UsiXML based User Interface method

Figure 4 shows an illustration of the forward engineering method. The detail of this method is fully explained in [5]. The starting point of the forward engineering is a task and a domain model (products). These models are then transformed (work) into an abstract UI (product) which is then transformed (work) into a concrete UI model (product). Finally the code (product) is generated (work). In order to achieve these transformations, a sequence of development steps (sequence of reifications and code generation) are performed. Each development step may involve a set of development sub-steps. For example, the first development step involves the development sub-step: "*Identification of Abstract UI structure*". This sub-step consists in the definition of groups of abstract interaction (an element of the abstract user interface). Each group corresponds to a group of tasks (in task model) tightly coupled together. To achieve its work, the sub-step can use a sequence of rules. For example, the sub-step: "*Identification of Abstract UI structure*" uses the sequence of two rules: R1 "For each leaf task of a task tree, create an Abstract Individual Element"; and R2: "create an Abstract Container structure similar to the task decomposition structure". And so on, each development step takes as input a UsiXML model(s) and transforms it (them) to another UsiXML model(s) by involving a set of development sub-steps, which in turn, manipulate sub-steps models by using a set of rules. Note that, a development sub-step can use templates of transformation instead of rules. For example, the step "generating the user interface code" can use a template based approach [7] in order to generate the UI code. Another note is that, each development step and development sub-step has a producer responsible of its execution. For example, the first development step can have a human actor who verifies the transformation done in this step. In turn, the sub-step: "*Identification of Abstract UI structure*" can have a transformation tool that can execute the rules sequence of this sub-step.

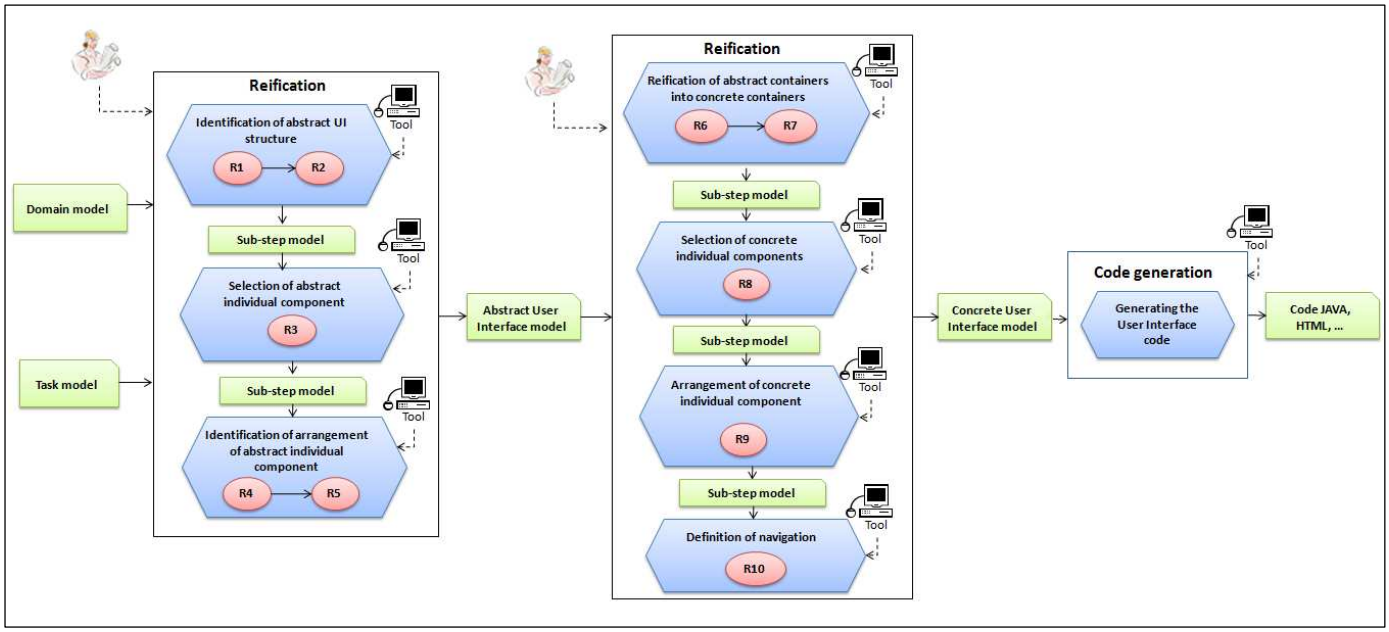

Figure 4. Forward Transformational Development of UIs

According to Henderson et al. in [8], a method meta-model needs to be generic enough so that any conceivable method can be expressed. In addition, a method meta-model needs to be concrete enough so that any methodological concepts can be treated with specific semantics. For this reason, the UsiXML method engineers need to rely on robust and well-defined meta-models. In the literature, several method meta-models have been introduced like SPEM [9], OPEN [10] and ISO 24744 [11]. We have conducted a comparative study of the method meta-models (see Appendix A). This study has shown that:

- SPEM 2.0 is an OMG standard. It reuses the UML diagrams to describe the elements of a method. This provides a great usability of this standard. In addition, SPEM separates the operational aspect of a method (Method Content), from the temporal aspect of a method (Process Structure). This allows using any modelling language to describe the process behaviour. However, the method engineer can define what elements will exist in the method layer, but characterizing endeavour layer elements (e.g. a specific project, organizational support activities, etc.) is not possible [12];
- OPEN provides a significant detail to describe the different elements of a method. However, like SPEM, the OPEN standard does not allow exerting control on the endeavour layer (e.g. a specific project, organizational support activities, etc.) from the meta-model layer [12]. In addition, the OPEN standard does not support the abstract generalization classes that allows to describe a customable method meta-model;
- ISO 24744 supports the dual-layer modelling that allows configuring the enactment of the method (endeavour layer) from the meta-model level by using the *Clabject* and the *Powertype* concepts (see Appendix A, Section A5). However, object-oriented programming languages (like JAVA) do not support the dual-layer [13, 14]. This is an issue since, in the UsiXML project, we plan to use Java based platforms (e.g. Eclipse, GMF, EMF, etc.) in order to develop the UsiXML support tools [15].

These standard meta-models can be adopted to describe the UsiXML development methodologies. But it is more suitable to have a specific method meta-model in order to support the specific key elements of the UsiXML development methodologies (e.g. development path, development step, and development sub-steps). For this reason, we propose in this document a new meta-model for the UsiXML development methodologies. The proposed meta-model is based on SPEM 2.0.This is justified by the following reasons:

- SPEM 2.0 provides a great usability, as well as, it is easier to implement since it is a UML profile;
- SPEM 2.0 contains abstract generalization classes (e.g. Kind element) for refining the vocabulary used to describe concepts or the relationship between concepts. These abstract generalization classes allow creating customable method meta-models specific to a certain domain (e.g. User Interface Development);
- SPEM 2.0 allows using any modelling language to describe the process behaviour. In particular, the BPMN standard can be used in order to automate the development process through a web services composition.

For this reason, the proposed meta-model is called SPEM4UsiXML. In the next section, we detail the proposed meta-model.

4. SPEM4USIXML META-MODEL

SPEM4UsiXML (SPEM for UsiXML) is dedicated to UsiXML method modelling. The goal of this meta-model is to propose the elements that are necessary to define any UsiXML method. SPEM4UsiXML extends SPEM 2.0 ([9]) by adding new classes. Therefore, SPEM4UsiXML (like SPEM) is a UML profile. In addition, SPEM4UsiXML (like SPEM) separates the operational aspect of a method from the temporal aspect of a method. This means that SPEM4UsiXML reuses the UML diagrams for the presentation of various UsiXML method concepts. As depicted in Figure 5, the SPEM4UsiXML meta-model uses seven main meta-model packages inherited from SPEM: *Method Content* describes the static aspect of a method; *Process Structure* and *Process Behaviour* describe the dynamic aspect of a 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 method; *Managed Content* describes the documentation of a method. Note that SPEM4UsiXML extends the classes of Method Content and Process Structure as we will explain thereafter.

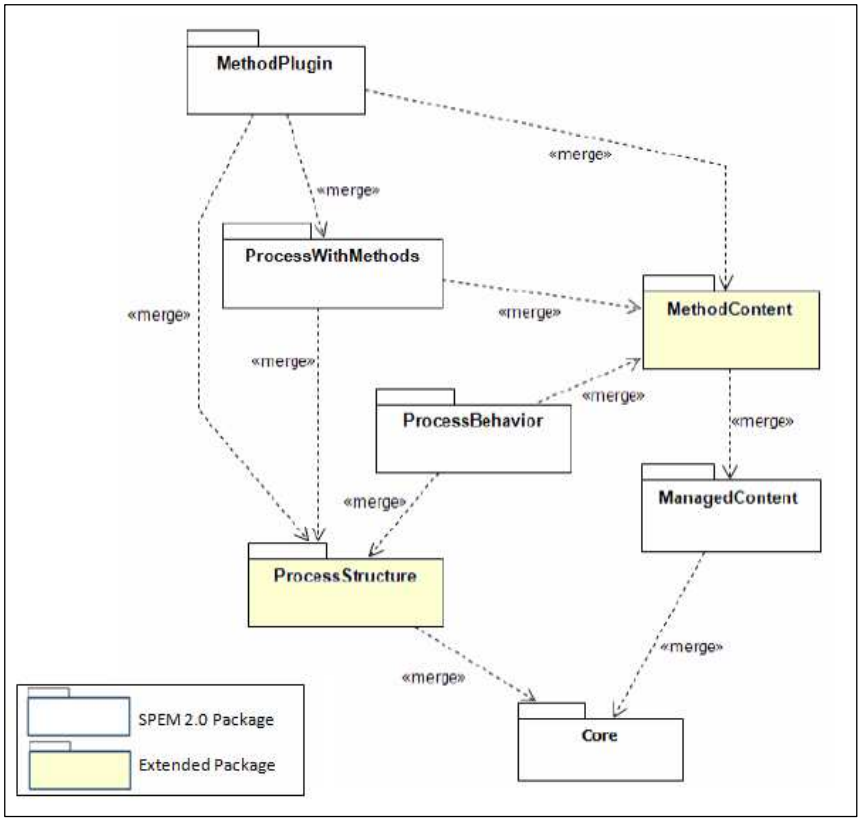

Figure 5. Structure of the SPEM4UsiXML meta-model

In the following we will detail the packages of the SPEM4UsiXML meta-model.

4.1. Core package

SPEM4UsiXML uses the SPEM 2.0 *Core* meta-model package. This package contains abstract generalization classes that are specialized in the other meta-model packages. These abstract generalization classes are used to define common properties of their specialized classes. For example, *Work Definition* is an abstract generalization class that represents the work being performed by a specific role, or the performed work throughout a lifecycle. It is used to define some default associations to *Work Definition Parameter* (e.g. owned Parameter) and *Constraint* (pre- and post-condition). Another example is the *Work Definition Parameter* that represents parameters for *Work Definitions*. *Work Definition Performer* is another example of the abstract generalization classes that represents the relationship of a work performer (role) to a *Work Definition*.

4.2. Method Content

The Method Content meta-model package defines the core elements of every method (producer, work unit, and work product) independently of any specific processes and development projects. In other words, the package defines how specific step development goals are achieved as well as

the involved roles, resources and results. However, the Method Content package does not specify the placement of these steps within a specific development lifecycle.

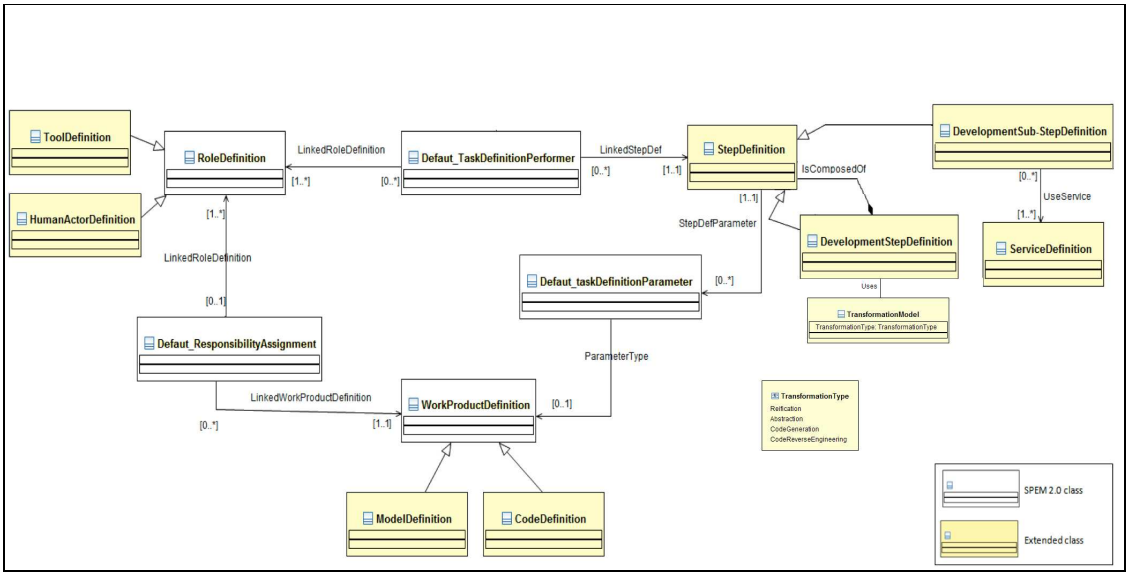

Figure 6. SPEM4UsiXML Method Content meta-model package

As shown in Figure 6, SPEM4UsiXML adds new classes to the original SPEM method content meta-model package in order to specify the several development steps and sub-steps and also the different kinds of product and producer. The important classes of the SPEM4UsiXML Method content meta-model are:

- *Development Step Definition:* defines the transformation being performed by *Roles Definition* instances. A *Development Step* is associated to an input(s) and an output(s) *Work Products*. A *Development Step Definition* can be:
    - *Reification Definition*: defines the transformation of a *Work Product Definition* of higher-level into a *Work Product Definition* of lower-level
    - *Abstraction Definition*: defines the transformation of a *Work Product Definition* of lower-level into a *Work Product Definition* of higher-level
    - *Translation Definition*: defines the transformation a *Work Product Definition* based on context
    - *Code generation Definition:* defines the transformation of *a Model Definition* into a *Code Definition*
    - *Code reverse engineering Definition:* defines the transformation of *a Code Definition* into a *Model Definition*
- *Development Sub-Step Definition:* defines the sub-steps of a *Development Step*. A sub-step can be achieved using a service (*Service Definition*). Each service can be based on a set of transformation rules, a program, the context or a template in order to enact the *Development Sub-Step.*
- *Step Definition*: is an abstract generalization class that defines a set of properties that are inherited by *Development Step,* and *Development Sub-Step*.

- *Work Product Definition:* describes the product which is used, modified, and produced by *Development Steps*. A Work Product Definition can be*:* a UsiXML model *(Model Definition)* or UI code *(Code Definition).*
- *Role Definition:* defines a set of related skills, competencies, and responsibilities of an individual or a set of individuals. Roles are used by *Development Step* or by *Development Sub-Step* to define who performs them as well as to define a set of *Work Product Definitions* they are responsible for. A *Role Definition* can be:
    - *Tool Definition:* describes any automation unit (e.g. CASE tool, or general purpose tool) that performs the *Development Step* or *Development Sub-Step.*
    - *Human Actor Definition*: describes any person, or organization that performs the *Development Step* or *Development Sub-Step.*

The SPEM4UsiXML Method Content meta-model contains also some useful elements inherited from SPEM 2.0, like:

- *Default Responsibility Assignment*: links *Role Definitions* to *Work Product Definitions*, by indicating that the *Role Definition* has a responsibility relationship with the Work Product Definition.
- *Default Step Definition Performer*: links *Role Definition* to *Development Step*, by indicating that the *Role Definition* instances participate in the work defined by the *Step Definition*.

Figure 7 shows a UML 2 use case diagram using a SPEM4UsiXML profile. This diagram represents a development step of the forward engineering instantiated from the Method Content meta-model package. The diagram presents the roles and products involved in the development step “Reification”. Indeed, this development step is executed by the primary role “Human Actor”. The task can be performed also by an additional role “Transformation Tool”. Finally, the diagram shows that all the input and output products (e.g. Task and domain models) are mandatory.

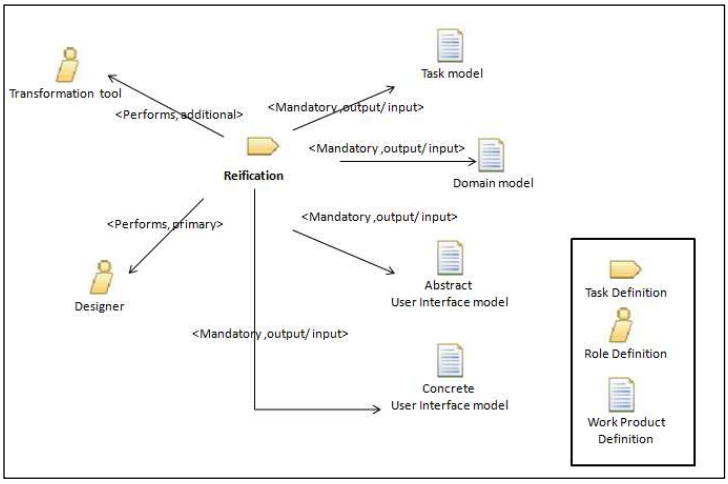

Figure 7. An example of the SPEM4UsiXML profile use case diagram of the a Reification definition

4.3. Process Structure

The Process Structure meta-model package defines the structure of the method process. This package represents a process through a static breakdown (decomposition) structure of *Development Step* classes that are linked to *Role* classes and *Work Product* classes.

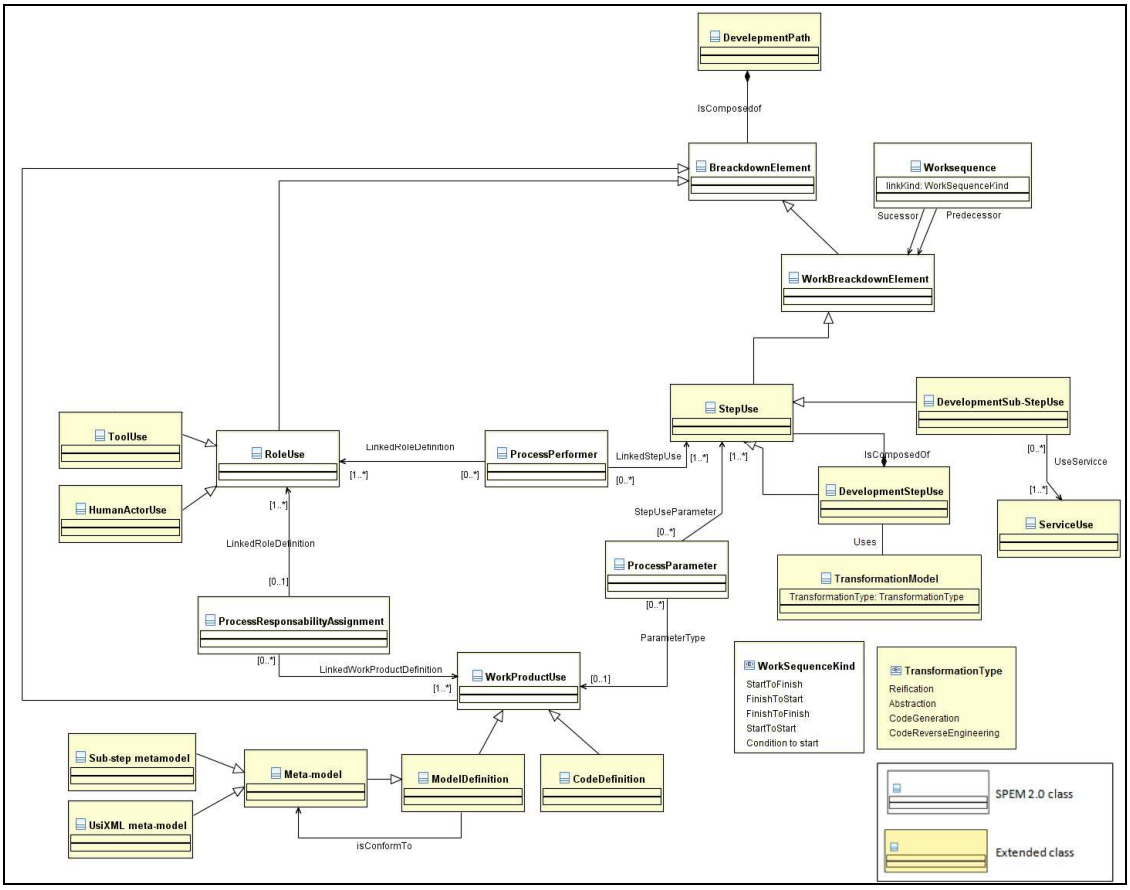

Figure 8. SPEM4UsiXML Process Structure package

As shown in Figure 8, 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.
- *Breakdown Element:* is an abstract generalization class that defines a set of properties available to the elements of a UsiXML method (Product, Development Step and Producer).
- *Work Breakdown Element:* provides specific properties for *Breakdown Elements* that represent *Development Step* and *Development Sub-Step*.
- *Step Use*: is an abstract generalization class that defines a set of properties available to *Development Step,* and *Development Sub-Step*.

- *Development Step Use:* defines the transformation steps of the method that are being performed by *Role Use* instances. A *Development Step Use* is associated to an input and an output *Work Product Use*. A *Development Step Use* can be: a reification (*Reification Use),* an abstraction (*Abstraction Use*), a translation (*Translation Use),* a code generation *(Code generation Use)* or a code reverse engineering (*Code reverse engineering Use*).
- *Development Sub-Step Use:* defines the sub-steps of a *Development Step Use*. As sub-step can be achieved using a service (*Service Use*).
- *Role Use:* represents a performer of a *Development Step Use* or a *Development Sub-Step Use*.
- *Work Product Use:* represents an input and/or output type for a *Development Step*. It can concern a model (*Model Use*) or code (*Code Use*).
- *Control Flow:* represents a relationship between two *Work Breakdown Elements* in which one *Work Breakdown Element* depends on the start or end of another *Work Breakdown Element* in order to begin or end.

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.

As explained above, the concepts of the Process Structure package represent a process as a static breakdown structure, by allowing to define predecessor dependencies amongst them, without defining the process modelling language that express the behaviour of the process. The latter is expressed separately in the Process Behaviour package.

4.4. Process Behaviour

The SPEM4UsiXML uses the SPEM 2.0 Process Behaviour meta-model package. This package allows extending these process structures with behavioural models. However, it does not introduce the formalism for enacting a method process. It rather proposes to reuse an existing externally-defined a behaviour model such as BPEL, UML 2 Activity diagram or BPMN (see Figure 9). [9] 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. Although, the separation provides a flexible way to represent the behavioural aspects of SPEM processes,

this package does not define the mapping rules to link the elements of SPEM process with the behavioural models. It rather proposes classes that help to define these mapping rules.

4.5. Process With Methods

As explained above, SPEM4UsiXML separates reusable core method content (expressed using the Method Content meta-model, see Section 4.2) from its application in processes (expressed using the Process Structure meta-model, see Section 4.3). The *Process With Methods* meta-model package allows integrating the process definition with instances of the core method content elements. This integration allows specifying how and which method elements will be applied in which part of the process. For example, a *Development Step Definition* (Section 4.2) can be invoked many times throughout a development path. Each invocation is defined with an individual element of the *Process With Methods* meta-model which is called *Development Step Use*. The *Process With Methods* meta-model package manages *Development Step* invocations by changing for example the roles involved in performing the task or an omission of specific work product input types. In other words, a *Development Step Use* represents a binding for a *Development Step Definition*. This is also valid for *Development Sub-Step Definition* and *Development Sub-Step Use.*

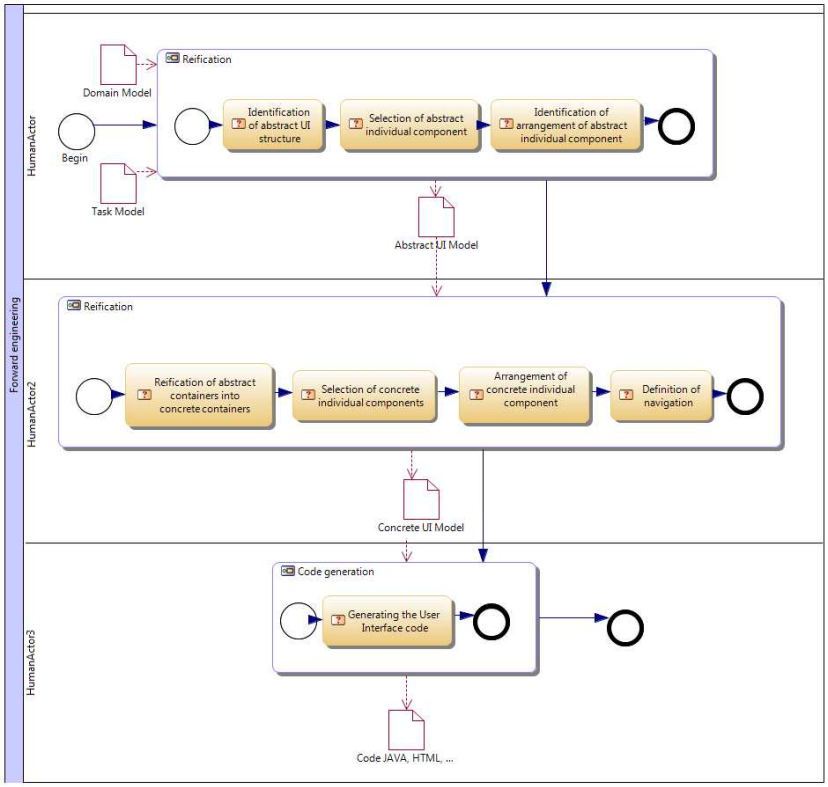

Figure 9. The process of the UsiXML forward engineering

Figure 9 represents a BPMN representation of the UsiXML forward engineering process. This process is composed of three sub-processes of activities related to the three development steps of the forward engineering development path. Each development step manipulates data objects that are related to the work product use (model or code). Finally, each development step (sub-process) is composed of a sequence of development sub-steps.

4.6. Managed Content

SPEM4UsiXML uses the SPEM 2.0 Managed Content meta-model package. This package introduces concepts for managing the textual documentation of a method. These concepts can be used independently (e.g. set of best practices) or they can be used in combination with the process structure by associating guidance elements with process structure elements.

4.7. Method Plug-in

SPEM4UsiXML uses the SPEM 2.0 Method Plug-in meta-model package. This package defines concepts for designing and managing repositories of method contents and processes. The concepts allow extending and personalizing the instance of Method Content and Process Structure by using plug-ins. This allows method configuration, where users select the process capabilities that are appropriate for their specific needs.

5. USIXML METHOD ENACTEMENT

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 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. 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. The motivation behind this separation is to give a method designer option to choose process behavior models that fits his/her needs. But, as explained above, SPEM does not define the mapping rules to link its elements process with the behavioral models. To deal with this limit, a set of mapping rules should be defined to map SPEM4UsiXML model to an enactment model.

According to The UsiXML FPP [6], 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 enacted by using a BPEL engine. For this reason, a set of mapping rules should be defined in order to transform, in the deployment-time, the elements of SPEM4UsiXML process with the OASIS standard BPEL [16]. In light of this, we will propose, in the deliverable of Task 2.4, a set of mapping rules used by a BPEL transformation tool to map a subset of SPEM4UsiXML concepts and the BPEL language.

6. CONCLUSION AND FUTURE WORK

In this document, we proposed a new meta-model for UsiXML method description that is called SPEM4UsiXML. This meta-model is based on the standard SPEM 2.0 which uses a UML profile to define the 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 performed by using a set of rules, a program, or a set of templates encapsulated within a service. SPEM4UsiXML separates the structural aspect of a method (Method Content) from the dynamic aspect of a method (Process Structure). This allows using any modelling language to describe the process behaviour like BPEL. Unfortunately, the SPEM4UsiXML meta-model cannot support the enactment of a UsiXML method on a specific endeavour. To deal with this limit, we plan to propose, in the deliverable of Task 2.4, a software architecture for supporting UsiXML methods that allows to transform 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.

7. REFERENCE

1. UsiXML, (2010), User Interface eXtensible Markup Language, Available online: http://www.usixml.org.
2. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J., (2003), A Unifying Reference Framework for Multi-Target User Interfaces, Interacting with Computers, 15(3), June 2003, pp.289–308UsiXML, (2010), User Interface eXtensible Markup Language, Available online: http://www.usixml.org
3. UsiXML Project, (2011), T1.3: UsiXML definition (Elicitation of the models requirements, contents, semantics, abstract and concrete syntaxes, and stylistics). In the deliverable of the Work package 1: Task 1.3, February 2011.
4. Stanciulescu, A., (2008), A Method for Developing Multimodal User Interfaces of Information System, Ph.D. thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, 25 June 2008
5. Q. Limbourg, J. Vanderdonckt, 2009. “Multipath Transformational Development of User Interfaces with Graph Transformations”. In Human-Centered Software Engineering, Human-Computer Interaction Series, Volume . ISBN 978-1-84800-906-6. Springer London, 2009, p. 107
6. ITEA2, (2009), UsiXML Full Project Proposal, December 23, 2009.
7. Czarnecki K, Eisenecker UW (2000) Generative Programming. Methods, Tools, and Applications, Addison-Wesley, Reading.

8. B. Henderson-Sellers and C. Gonzalez-Perez, 2005, “Metamodelling for Software Engineering”. ISBN-13: 978-0470030363Wiley (October 14, 2008)
9. OMG, 2008 “Software & Systems Process Engineering Meta-Model Specification version 2.0”, In OMG Document Number: formal/08-04-02. Standard document URL: http://www.omg.org/spec/SPEM/2.0/PDF
10. International Organization for Standardization / International Electrotechnical Commission, 2007. “ISO/IEC 24744. Software Engineering - Metamodel for Development Methodologies”, JTC 1/SC 7, 2007
11. OPEN Consortium, 2010, “OEPN”, http://www.open.org.au/
12. B. Henderson-Sellers and C. Gonzalez-Perez, 2005, “A comparison of four process metamodels and the creation of a new generic standard”. Information and Software Technology. Volume 47, Issue 1, 1 January 2005, Pages 49-65
13. T. Kuhne; D. Schreiber; 207, “Can Programming be Liberated from the Two-Level Style?: Multi-Level Programming with DeepJava” in OOPSLA'07 International Conference on Object-Oriented Programming, Systems, Languages, & Applications No22, Montréal , CANADA (21/10/2007)
14. M. Gutheil, B. Kennel, C. Atkinson, 2008, “A Systematic Approach to Connectors in a Multi-level Modeling Environment”. Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems (MoDELS 2008): 843-857
15. Defimedia, 2010, “UsiXML Software tools requirements specification”. In the deliverable of the Work package 3: Task 3.1, September 2010
16. 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)

A. APPENDIX: COMPARATIVE ANALYSIS OF META-MODELS FOR DEVELOPMENT METHODS

In this appendix, we propose a comparative study of method meta-model standards in order to select the most appropriate meta-model for the UI method development based on UsiXML language.

A1. INTRODUCTION

The task 2.1 of the UsiXML project aims to provide a unified software engineering method for User Interface based on UsiXML language. This method describes the process to follow during a User Interface design. The method needs to be compliant with a well-defined meta-model so that we formally define the core elements of the UsiXML method.

In the literature, three major method meta-model standards have been proposed: SPEM 2.0 [OMG 2008], OPEN [OPF 2005] and ISO 24744 [ISO 2007]. These standards describe, in different ways, the core elements of a method (work unit, work product, and producer). Each standard is built on different main principles:

- SPEM [OMG 2008] separates the operational aspect of a method from the temporal aspect of a method;
- OPEN [OPF 2005] defines an industry-standard for software method modelling;
- ISO 24744 [ISO 2007] uses a dual-layer modelling to allow the method engineer to configure the enactment of the method from the meta-model level.

In this document, we propose a comparative study of these three meta-model standards in order to select the most appropriate meta-model for the UI method development based on UsiXML language.

The rest of this appendix is organized as follows. In Section A2, we describe the different approaches for the collaborative design of meta-models. In Section A3, Section A4 and Section A5 we describe respectively the SPEM 2.0 standard, OPEN standard, and ISO 24744 standard. We conducted a comparative study of these approaches based on different comparison criteria. In Section A6 we present the comparative study of the method meta-model standards.

A2. BACKGROUND

In this section, background definitions for method and meta-modelling for development methods[1] are given. *A method is a systematic way of doing things in a particular discipline* [ISO 2007]. It specifies the process to follow together with the work products to be used and generated by the involved people and tools, during a development effort [Hen 2008]. According to this definition, three major aspects of a method can be considered (Figure A.1):

- *The process aspect* represents the work that must be done. It is defined in terms of tasks, steps, activities, etc. This process is usually called **Work Unit**.
- *The product aspect* represents the artefact that must be manipulated (i.e. created, used or changed). It can concerns model, document, hardware, software, etc. These artefacts are usually called **Work Products**.

[1] In this document we consider that the terms "*methodology*" and "*method*" are synonymous [Hen 2008]

- *The organization aspect* represents the agent that has the responsibility to execute a work unit. It is defined in terms of person, role, team, tool, etc. This agent is usually called **Producer**.

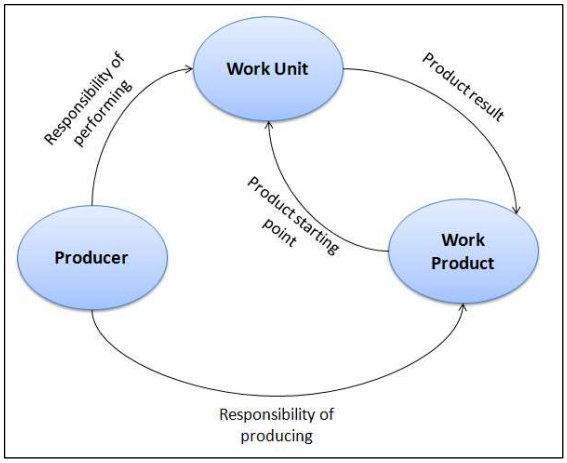

Figure A.1. The major aspects of a method [Hen 2008]

Figure A.2 represents an example of a development method of the User Interface (UI). The starting point of this UI development method is the construction of a task model and a domain model by a designer. These two models are then transformed into an abstract UI model which is then transformed into a concrete UI model by a transformation tool. Finally, the concrete UI model is used to generate UI code.

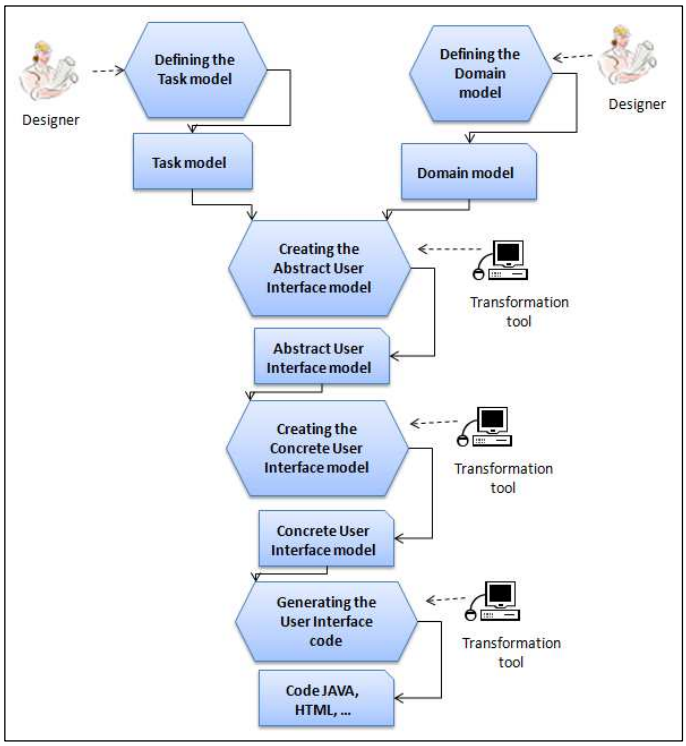

Figure A.2. An example of the User Interface development method [Lim 2009]

To be useful, a method needs to be designed and evaluated by describing formally its content (the semantics of the method) and its form (the abstract/concrete syntax of the method). In addition, a method and its enactment need to be supported by tools. The content of a method refers to the prescription of the process to follow, the work products to be used and the responsible of the work performed in a particular domain. The form of a method refers to the expression of the three major aspects of a method (work unit, work product, and producer) and the relationships between them. In this document, we focus on the form of the method. Indeed, the method must be expressed using a specialized modelling language, so that, minimizing ambiguity, the method can easily be processed by a computer. For this reason, the meta-modelling approach can be used in order to deal with the expression of method. From the point of view of a meta-modelling approach, a method is seen as a model of the future scenario of its enactment. In addition, a method meta-model defines an explicit description of how the method model is built.

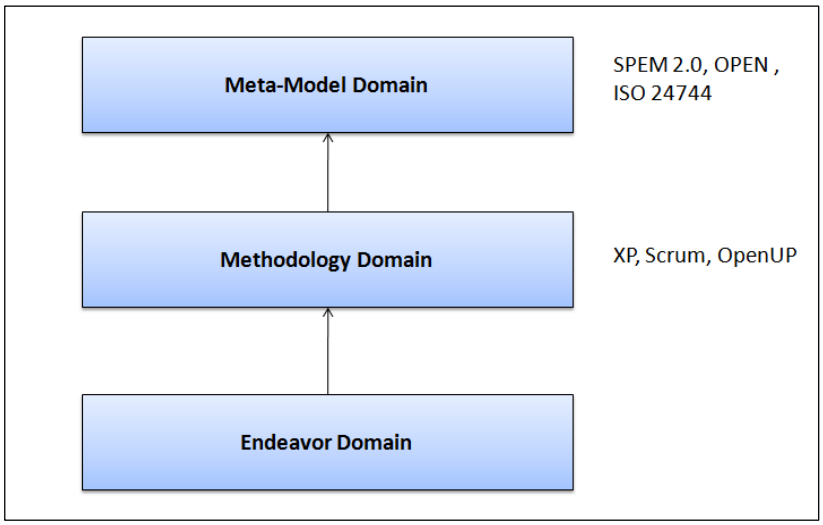

Figure A.3. Three levels of the method meta-modelling

As directed in Figure A.3, the meta-modelling approach describes a development method in three levels of abstraction (see Figure A.3).

- *The meta-model level* defines a meta-model that describes how the method model can be constructed by using a set of classes to formally specify the concepts of the method and the relationships between them. According to Henderson et al. in [Hen 2008], a method meta-model needs to be generic enough so that any conceivable method can be expressed. In addition, method meta-model needs to be concrete enough so that any methodological concepts can be treated with specific semantics. For this reason, the method engineers need to rely on robust and well defined meta-models. In the literature, several method meta-models have been introduced like SPEM [OMG 2008], OPEN [OPF 2005] and ISO 24744 [ISO 2007]. In this document, we will focus on this level by conducting a comparison study of the method meta-models.
- *The model level* defines the method model that describes the prescription of the work units to perform, the work products to be manipulated and the agents who have the responsibility of performing the work (e.g. XP [XP 2010], Scrum [Scrum 2010] and OpenUP [OpenUP 2010]). The method model classes are obtained by creating instances

The meta-modelling approach for development methods offers a flexible way to describe a method. The purpose of this approach is to use a method meta-model to increase productivity of method engineers and to improve the quality of the method models. As explained above, three major method meta-model standards have been proposed: SPEM [OMG 2008], OPEN [OPF 2005] and ISO 24744 [ISO 2007]). These standards describe, in different ways, the core elements of a method (work unit, work product, and producer). In the following sections, we detail these standards.

A3. SPEM 2.0 META-MODEL SPECIFICATION

The SPEM (Software & Systems Process Engineering Meta-Model) [OMG 2008] is an OMG standard dedicated to software method modelling. The goal of SPEM is to propose minimal elements necessary to define any software and systems development method, without adding specific features to address particular domains. As a result, this meta-model supports a large range of development methods of different styles, levels of formalism, and lifecycle models.

SPEM is a UML profile. This means that SPEM reuses the UML wherever possible. Consequently, the SPEM uses the UML profile for the presentation of various software method concepts.

The current version of SPEM (version 2.0 [OMG 2008]) was completely reformulated (from SPEM 1.0 [OMG 2002]) in order to separate the operational aspect of a method from the temporal aspect of a method. As depicted in Figure A.4, the SPEM 2.0 meta-model uses seven main meta-model packages: *Method Content* package describes the static aspect of a method; *Process Structure* and *Process Behaviour* packages describe the dynamic aspect of a method, *Process With Methods* package describes the link between these two aspects; *Core* package provides the common classes that are used in the different packages; *Method Plug-in* package describes the configuration of a method and *Managed Content* package describes the documentation of a method.

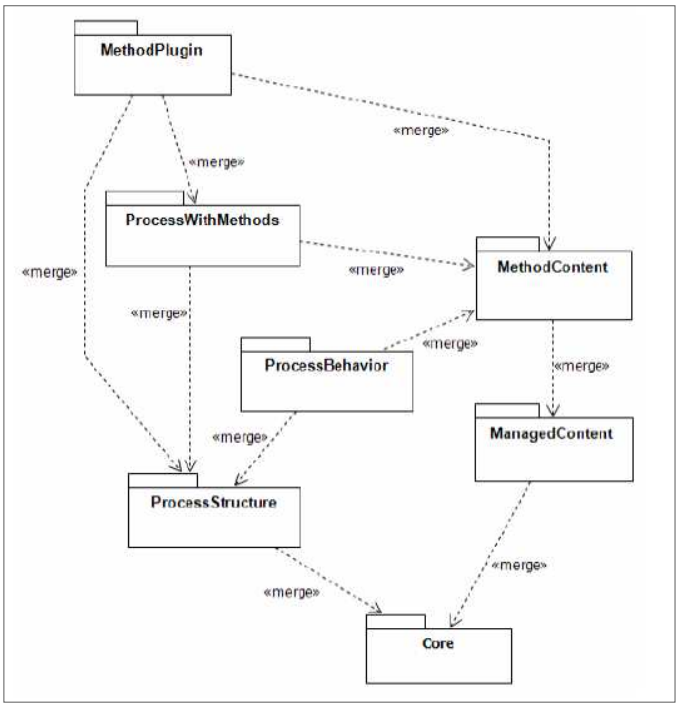

Figure A.4. Structure of the SPEM 2.0 meta-model [OMG 2008]

In the following, we detail the SPEM 2.0 meta-model packages

A3.1. Core package

The *Core* meta-model package contains abstract generalization classes that are specialized in the other meta-model packages. These abstract generalization classes are used to define common properties of their specialized classes. The *Core* meta-model mainly defines several important elements like:

- The *Kind* class that expresses a refined vocabulary specific to a method (e.g. Phase, Iteration, and Increment can be used as a kind for *Breakdown Elements* (see Section A5.3).
- Three abstract generalization classes that define the common properties of the three key concepts of a method: work unit, product and producer.
    - *Work Definition* is an abstract generalization class that represents the work being performed by a specific role, or the work performed throughout a lifecycle. It is used to define some default associations to *Work Definition Parameter* (e.g. owned Parameter) and *Constraint* (pre- and post-condition).
    - *Work Definition Parameter* is an abstract generalization class that represents parameters for *Work Definitions*.
    - *Work Definition Performer* is an abstract generalization class that represents the relationship of a work performer (role) to a *Work Definition*.

The *Core* elements are principally specialized in the packages *Method Content* and *Process Structure*.

A3.2. Method Content

The Method Content meta-model package defines the core elements of every method (producer, work unit, and work product) independently of any specific processes and development projects. In other words, describes the specific development steps that are achieved by which roles with which resources and results, without specifying the placement of these steps within a specific development lifecycle.

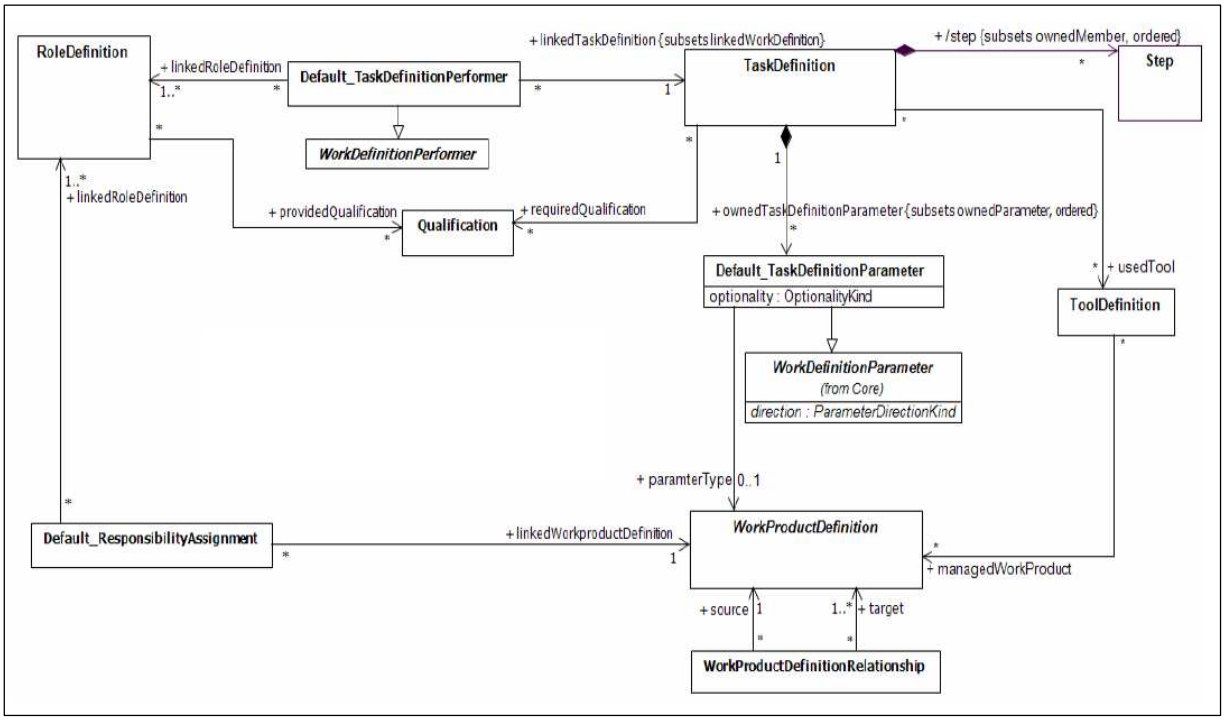

Figure A.5. SPEM 2.0 Method Content meta-model package [OMG 2008]

Figure A.5 illustrates the Method Content meta-model package. The main classes of the Method content meta-model are:

- *Task Definition:* defines the work being performed by *Roles Definition* instances. A *Task* is associated to input and output *Work Products*.
- *Step:* describes a meaningful and consistent part of the overall work described for a *Task Definition*. The collection of *Steps* defined for a *Task Definition* represents all the work that should be done to achieve the overall development goal of the *Task Definition*.
- *Work Product Definition:* describes the product which is used, modified, and produced by *Task Definitions*.
- *Role Definition:* designs a general reusable definition of an organizational role. It defines a set of related skills, competencies, and responsibilities of an individual or a set of individuals. Roles are used by *Task Definitions* to define who performs them as well as to define a set of *Work Product Definitions* they are responsible for.

The Method Content meta-model contains also useful elements like:

- Three relationships between the core method concepts can be expressed. The type of these relationships needs to be defined by using kind class instances (Section A3.1).
  - *Default Responsibility Assignment*: links *Role Definitions* to *Work Product Definitions*, by indicating that the *Role Definition* has a responsibility relationship with the Work Product Definition.
  - *Work Product Definition Relationship*: expresses a general relationship among *Work Products Definitions*.
  - *Default Task Definition Performer*: links *Role Definition* to *Task Definitions*, by indicating that the *Role Definition* instances participate in the work defined by the *Task Definition*.
- Additional classes design special elements like:
  - *Tool Definition:* describes any automation unit (e.g. CASE tool, or general purpose tool) that supports the associated instances of *Role Definitions* in performing the work defined by a *Task Definition*. A *Tool Definition* can identify a resource as useful, recommended, or necessary for a task's completion.
  - *Default Task Definition Parameter*: represents a special *Work Definition Parameter* that uses *Work Product Definitions* as well as adds an optionally attribute.
  - *Qualification:* documents the required qualifications, skills, or competencies for Role and/or Task Definitions.

Figure A.6 shows a UML 2 use case diagram using a SPEM 2.0 profile. This diagram represents a method model instantiated from the Method Content meta-model package. This model designs the UI development method which is presented in Section A2. The model of Figure A.6 presents the roles and products involved in the task "Creating the Abstract User Interface model". Indeed, this task is executed by the primary role "Transformation tool". The task can be performed also by an additional role "designer". The model shows that all the input products (Task and domain models) are mandatory. Finally, the task "Creating the Abstract User Interface model" returns the output product "Abstract User Interface model".

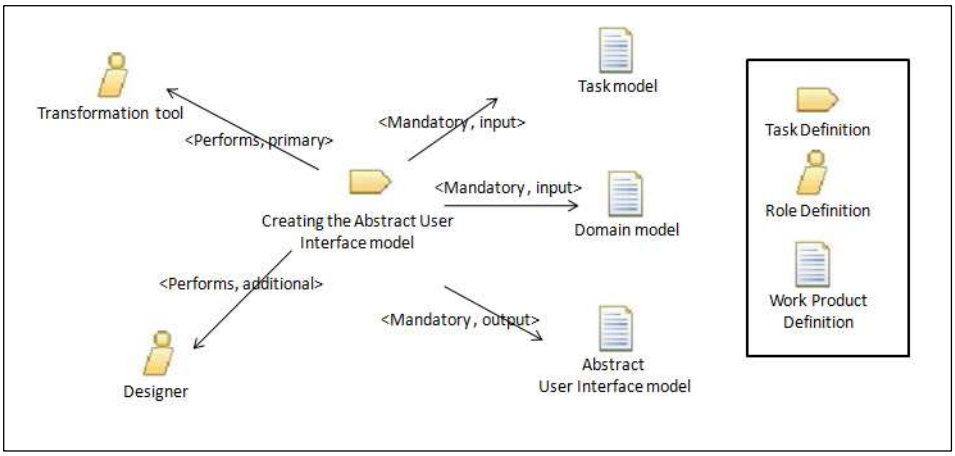

Figure A.6. An example of the SPEM 2.0 profile use case diagram of a task definition

A3.3. Process Structure

The Process Structure meta-model package defines the structure of the method process. This package represents a process through a static breakdown (decomposition) structure of *Activity* classes that are linked to *Role* classes and *Work Product* classes. This structure is useful to express for example the fact that a life-cycle is composed by set of phases, and each phase is composed by set of activities.

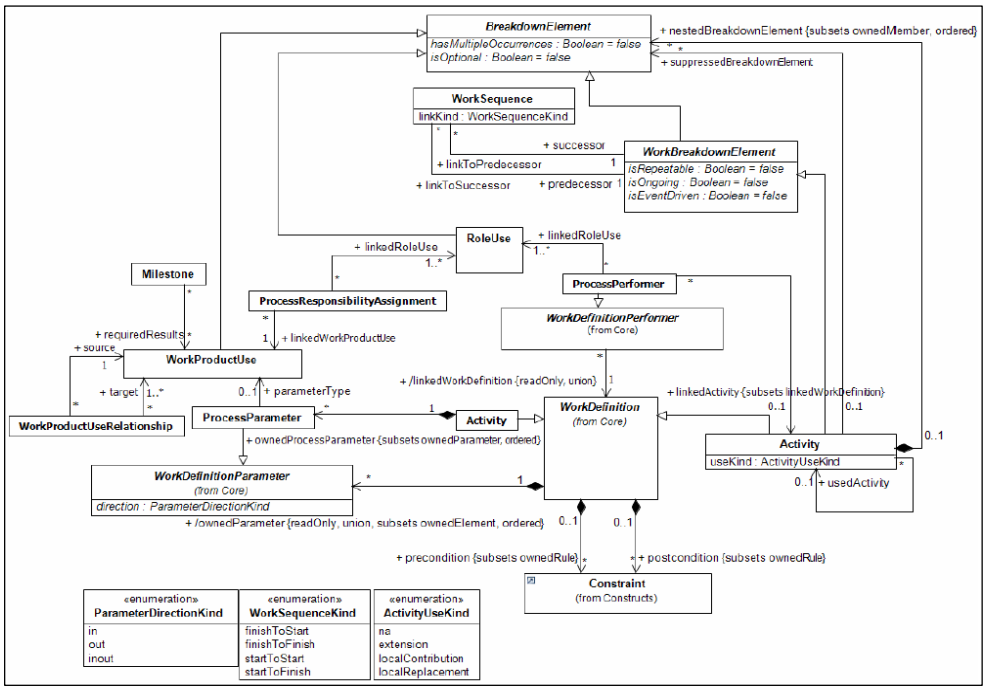

Figure A.7. Overview of the main classes and associations of Process Structure package [OMG 2008]

Figure A.7 illustrates the Process Structure meta-model package. The important classes of the Process Structure meta-model are:

- A *WorkDefinition* (coming from the Core package) is performed by a *Work Definition Performer*, which is a role, and, through this role, by a process performer. It can be submitted to *Constraints*, i.e. pre and post-conditions.
- *Breakdown Element:* is an abstract generalization class that defines a set of properties available to all of its specializations.
- *Work Breakdown Element:* provides specific properties for *Breakdown Elements* that represent work.
- *Activity:* defines basic units of work within a process as well as a process itself. In other words, every activity can represent a process in SPEM 2.0. It relates to *Work Product Use* instances via instances of the *Process Parameter* class and *Role Use* instances via *Process Performer* instances. An activity can be used by another activity (by using the *usedActivity* relationship), so that the structure of the source activity is copied into the target activity. This copy can be modified or completed.

- *Role Use:* represents a performer of an *Activity* or a participant of the *Activity*. A *Role Use* is only specific to the context of an Activity. It is not a general reusable definition of an organizational role like the *Role Definition* of the Method Content package (Section A3.2). Here too, a role has responsibilities on a product (ProcessResponsibilityAssignment).
- *Work Product Use:* represents an input and/or output type for an *Activity* or represents a general participant of the *Activity.*
- *Work Sequence:* represents a relationship between two *Work Breakdown Elements* in which one *Work Breakdown Element* depends on the start or finish of another Work *Breakdown Elements* in order to begin or end. The attribute *linkKind* refers to the enumeration *WorkSequenceKind* that defines four types of sequences between two *Work Breakdown Elements*.

The Process Structure meta-model contains also useful elements like:

- Three relationships between the core method concepts can be expressed. The type of these relationships needs to be defined by using kind class instances (Section A3.1).
  - *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.*
  - *Work Product Use Relationship:* expresses a general relationship among work products.
  - *Process Performer:* links *Role Uses* to *Activities* by indicating that these *Role Use* instances participate in the work defined by the activity.
- Additional classes design special elements like:
  - *Activity Use Kind:* provides mechanisms for dynamically linking Activities for reuse to other Activities or Processes
  - *Process Parameter:* defines input and output meta-types to be *Work Product Uses*.
  - *Milestone:* describes a significant event in a development project, such as a major decision, completion of a deliverable, or meeting of a major dependency (like completion of a project phase.

However, the concepts of the Process Structure package represent a process as a static breakdown structure, by allowing defining predecessor dependencies amongst them, without defining the process modelling language that express the behaviour of the process. The latter is expressed separately in the Process Behaviour package.

A3.4. Process Behaviour

The Process Behaviour meta-model package allows extending these process structures with behavioural models. However, it does not introduce its own formalism for behaviour models, but instead provides 'links' to existing externally-defined behaviour models, enabling reuse of these approaches from other OMG or third party specifications. For example, a process defined with the Process Structure concepts can be linked to UML 2 Activity diagram or BPMN that allow representing a process or to UML 2 State Machine diagram that allows representing the product states (see Figure A.8).

A3.5. Process With Methods

As explained above, SPEM 2.0 separates reusable core method contents (expressed using the Method Content meta-model, see Section A3.2) from its temporal aspect (expressed using the Process Structure meta-model, see Section A3.3). The *Process With Methods* meta-model package allows integrating the process definition with instances of the core method content elements. This integration allows specifying how and which method elements will be applied in a particular part of the process. For example, a *Task Definition* (Section A3.2) can be invoked many times throughout a development process. Each invocation is defined with an individual element of the *Process With Methods* meta-model which is called *Task Use*. The latter manages the *Task Definition* invocation by changing for example the roles involved in performing the task or an omission of specific work product input types. In other words, a *Task Use* represents a binding for a *Task Definition* in the context of one specific *Activity*. Therefore, one *Task Definition* can be represented by *many Task Uses;* each within the context of an *Activity* with its own set of relationships [OMG 2008].

Figure A.8 represents the life-cycle of the User Interface development method. The life-cycle is composed of a set of activities: Firstly, two activities of the definition of the task model and the domain model are performed in parallel way. Secondly, the transformation activities of the abstract UI model and the concrete UI model are performed in sequence way. Finally, generating UI code activity is performed. Each activity of this life-cycle is an element of type Task Use which reuses the element of type Task Definition. Note that the method process behaviour is expressed using the BPMN (see Section A3.4).

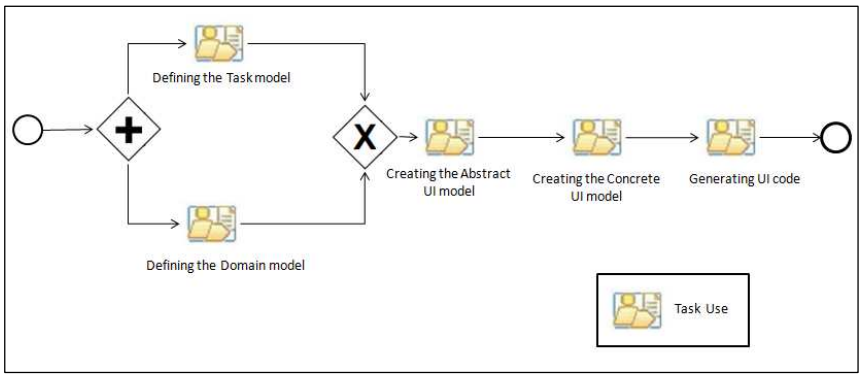

Figure A.8. The life-cycle of the User Interface development method

A3.6. Managed Content

The Managed Content meta-model package introduces concepts for managing the textual documentation of a method. These concepts can be used independently (e.g. set of best practices) or can be used in combination with the process structure, by associating guidance elements with process structure elements.

A3.7. Method Plug-in

The Method Plug-in meta-model package defines concepts for designing and managing repositories of method content and processes. These concepts allow extending and personalizing

the instances of Method Content and Process Structure by using plug-ins. This allows method configuration, where users select the process capabilities that are appropriate for their specific needs.

Note finally that SPEM 2.0 is now stable and several editors have adopted it in their tools. For example, Eclipse Foundation hosts a project that aims at providing an extensible framework for software method engineering [EPF 2010]. This project is called Eclipse Process Framework (EPF) and allows offering method and process authoring, library management, configuring and publishing a process. In addition, EPF proposes several method libraries (Method Plug-in, see Section A3.7) like XP [XP 2010], Scrum [Scrum 2010] and OpenUP [OpenUP 2010]. Figure A.9 depicts the interface of the EPF in which we can define the core elements of a method. Another example is the No Magic company that adopts the SPEM 2.0 to specify software method engineering in its commercial CASE tool called MagicDraw [NoMagic 2010].

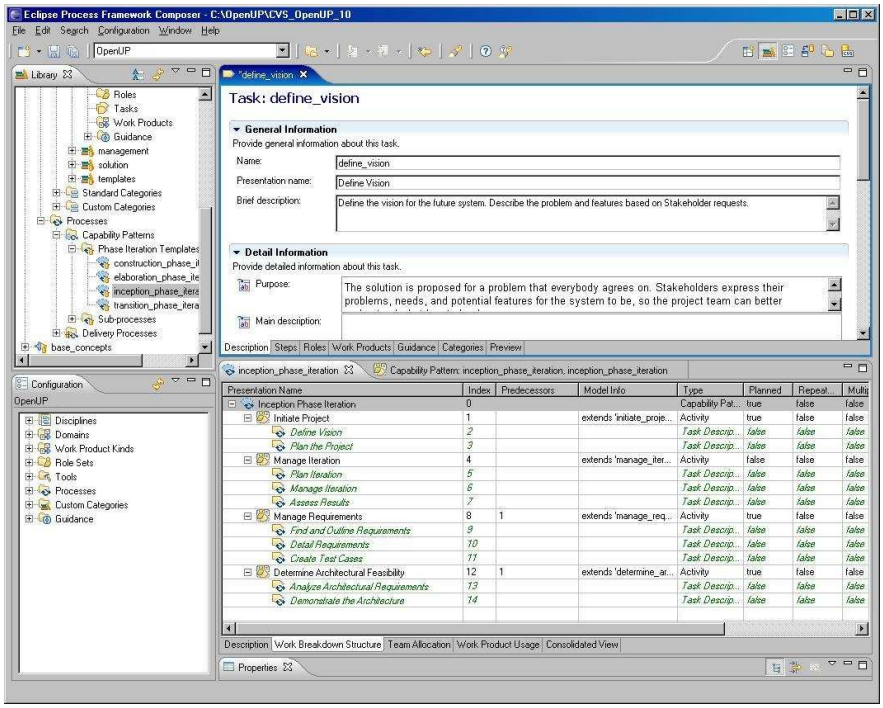

Figure A.9. The interface of the Eclipse Process Framework

A4. OPEN META-MODEL

Object-oriented Process, Environment, and Notation (OPEN) is an industry-standard for software method modelling [OPF 2005]. OPEN was developed and is maintained by an international consortium, which regroups several experts in method, universities, CASE tool vendors and developers [OPEN 2010]. As illustrated in Figure A.10, the OPEN meta-model uses six main meta-model packages to describe: the element that is developed during a project of a method (*Work Product*), the element that produces a work product (*Producer*), the element that is

performed by producers when developing work products (*Work Unit*), the time interval of work units (*Stage*), the element that is used to document work products (*Language*), and the organization or project under which the producer performs the work unit (*Endeavour*).

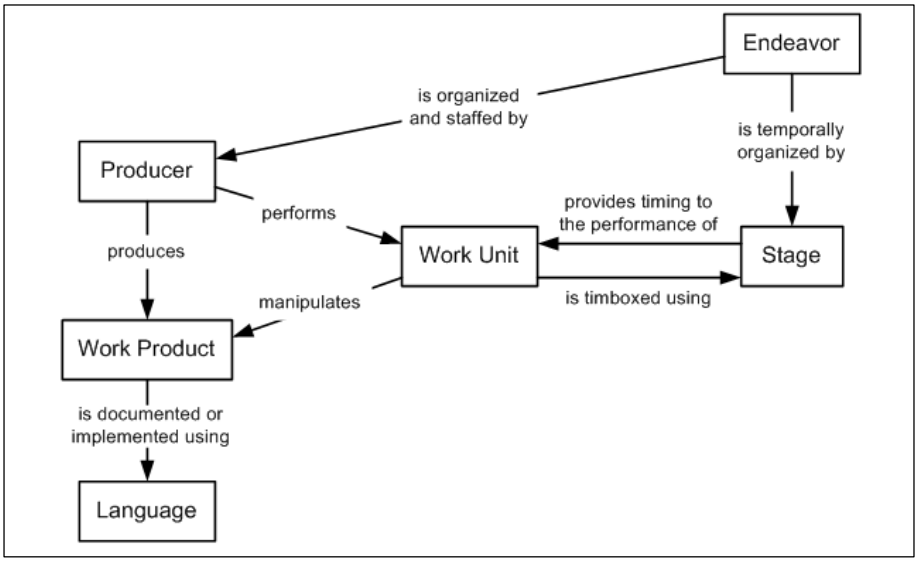

Figure A.10. OPEN Meta-Model packages [OPF 2005]

In the following, we detail the OPEN meta-model packages.

A4.1. Producer and Endeavour

A Producer is someone or something that performs *Work Units* and produces (i.e., creates, evaluates, iterates, or maintains), either directly or indirectly, versions of one or more *Work Products* (see Figure A.11):

- *Direct Producers*: consists of persons as well as roles played by the people and tools that they use;
- *Indirect Producers*: consists of teams of people (the membership of teams strictly being roles), organizations (the membership of which are teams) and endeavours.

An Endeavour is staffed by one or more organizations. It may be classified as projects, programmers or enterprises.

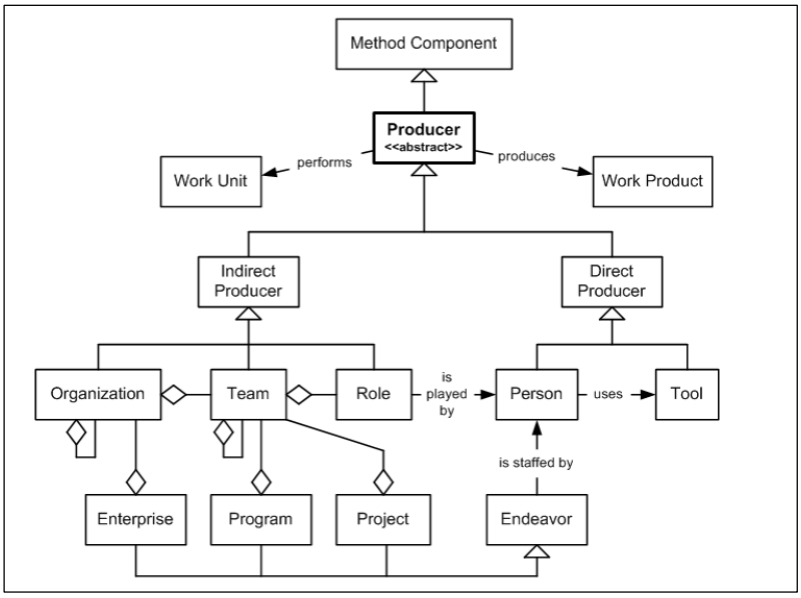

Figure A.11. OPEN Producer Meta-Model Package [OPF 2005]

A4.2. Work Products

A *Work Product* models anything that is produced, used, modified, or destroyed during the performance of one or more *Work Units* by one or more collaborating *Producers*. A *work product* is any significant thing of value (e.g., document, diagram, model, class, application, etc.) that is developed during a project (see Figure A.12). The *Work Product* meta-model defines also two important elements:

- *Work Product Set:* a set of work products (e.g., products produced by the tasks of a single activity)
- *Work Product Version:* a unique identification of a work product at a specific point in time that is created during an incremental or iterative development process

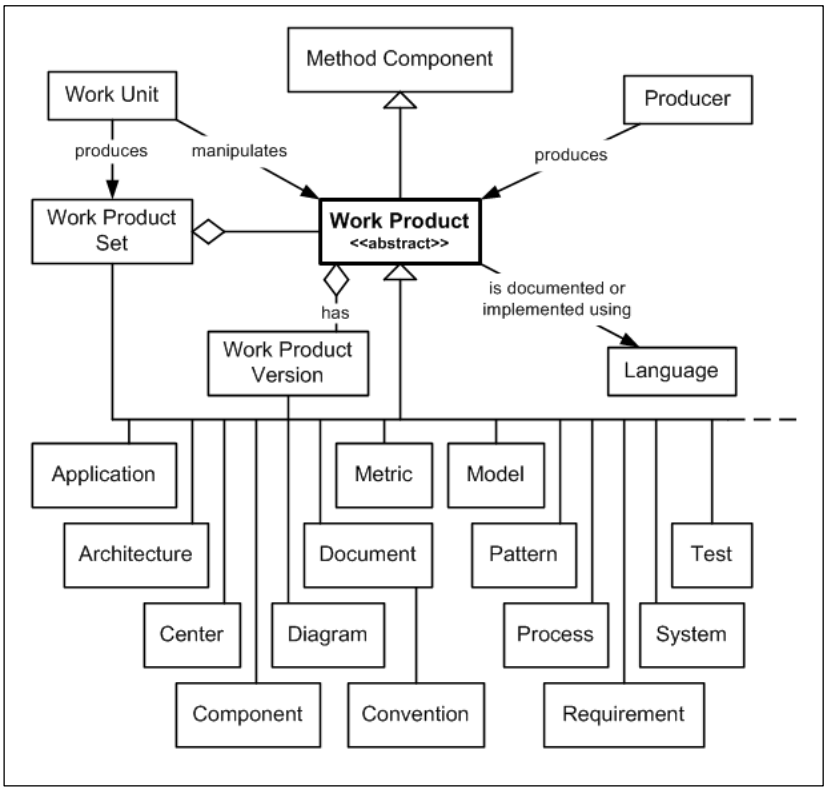

Figure A.12. OPEN Work Product Meta-Model Package [OPF 2005]

A4.3. Work Units

A *Work Unit* models a functional operation that may be performed by one or more *Producers* as part of one or more endeavour-specific processes. The *Work Unit* package is refined by up to 300 sub-classes. Figure A.13, depicts important elements of the *Work Unit* meta-model like:

- *Activity*: consists of a collection of tasks that produce a related set of *Work Products*.
- *Work Flow*: consists of a collection of tasks that produce a single work product.
- *Task*: consists of a single assigned job that may be performed by one or more *Producers*.
- *Technique*: models a way of performing a task (i.e., an implementation of a task using the Strategy Pattern).

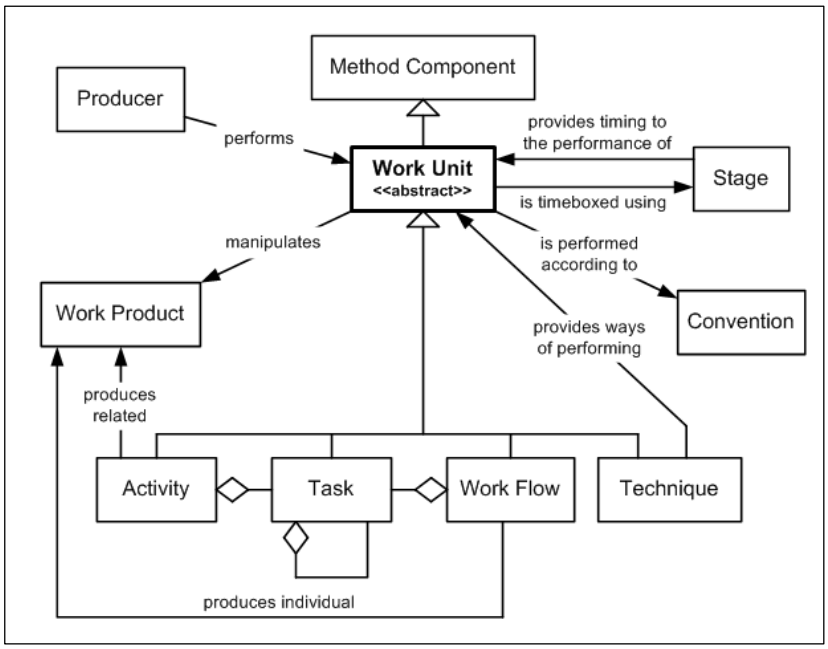

Figure A.13. OPEN Work Unit Meta-Model Package [OPF 2005]

A4.4. Stages

A *Stage* models the intended timing of the performance of a set of work units during the enactment of a method. As depicted in Figure A.14, the *Stage* meta-model defines several important elements like:

- *Stage with Duration:* models a period of time during which one or more work units are to be performed. It is part of the following inheritance elements:
  - *Cycle:* consists of one or more related phases. It can concern a *Life-Cycle* during which a single product is produced, used, and retired or a *Development Cycle* during which a single product is developed and delivered.
  - *Phase:* is a major logical partition of a cycle.
  - *Build:* is a component part of a phase.
- *Stage without Duration:* models a point in time in which one or more *Work Products* are to be produced or in which one or more *Work Units* are to be performed. It is part of the following inheritance elements:
  - *Milestone*: models a point in time during the delivery process in which a set of significant objectives is to be achieved (e.g., set of tasks completed, set of work products delivered).

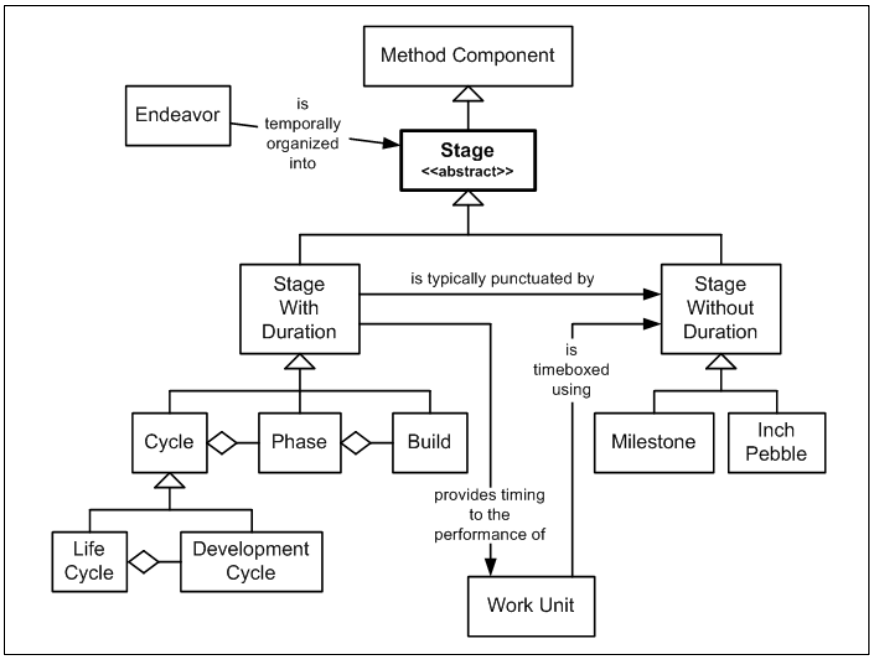

Figure A.14. OPEN Stage Meta-Model Package [OPF 2005]

A4.5. Languages

A *Language* consists of a vocabulary and a set of grammatical rules used to document a *Work Product*. For example, documents are mostly written in a natural language such as English and models are written using a modelling language such as UML.

Note finally that several commercial OPEN CASE tools are proposed like: eTrack tool [eTrack 2010], ArcStyler tool [ArcStyler 2010] and Myriad tool [Myriad 2010].

A5. ISO 24744 META-MODEL

ISO/IEC 24744 is an international standard meta-model for software method modelling [ISO 2007]. This standard, which is also called Software Engineering Meta-model for Development Methodologies (SEMDM) [Hen 2005], defines three major aspects to describe methodologies: the process to follow, the products to use and generate, and the people and tools involved. The standard proposes also graphical notations to represent concepts to help method engineers to easily design their methods.

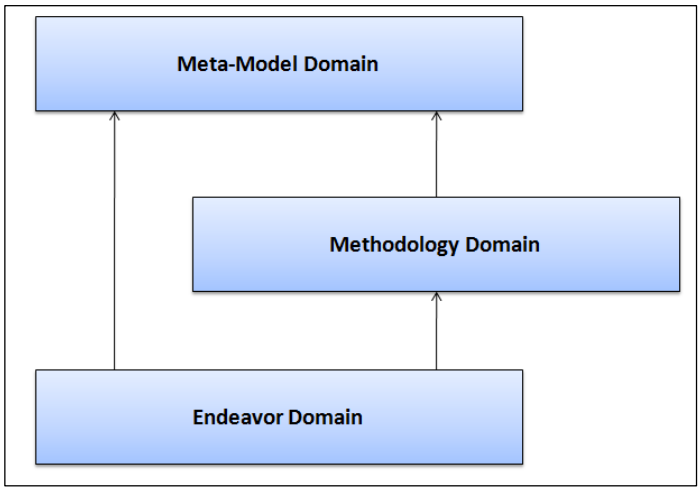

Figure A.15. Dual-Layer Modelling Approach of ISO 24744 [ISO 2007]

As explained in Section A2, the method meta-modelling approach supposes to use a meta-model to describe how the method model is built. The classes of the meta-model are instantiated (i.e. creation of objects), by method engineers, to generate a method model. In turn, classes of the method model are instantiated, by developers, to describe the method application in a specific endeavour. However, the instantiated objects in the method level are used as classes by developers to create elements in the endeavour level (i.e. the method enactment). Therefore, elements in the method level act as objects and classes at the same time. This apparent contradiction, not solved by any of the existing meta-modelling approaches [Hen 2005], is addressed by the ISO 24744 [ISO 2007]. For this reason, this standard uses a dual-layer modelling to allow the method engineer to configure the enactment of the method from the meta-model level (see Figure A.15). Indeed, the dual-layer modelling is based on two new modelling patterns [Hen 2005]:

- *PowerType pattern*: models two classes in which one of them represents the class "kinds" (called partitioned type class) of the other class (called a powertype class) [Ode 1994]. In order words, the pattern models the possibility to define each sub-kind (i.e. subtype) of a powertype class as a proper class (called partitioned type class) that should be defined in meta-model level. By convention, the powertype class takes the suffix "kind". For example, in the ISO 24744 meta-model, the Document class represents documents managed by developers (partitioned type class), while the DocumentKind class in the meta-model represents different kinds of documents that can be managed by developers (powertype class). In this Standard, the notation Document/*Kind is used to refer to the powertype pattern formed by the powertype DocumentKind and the partitioned type Document [ISO 2007].
- *Clabject pattern:* models a dual entity that is a class and an object at the same time [Atk 2000]. In other words, the pattern models a special element that has two facets in the same level: a class facet that contain typed attributes and an object facet that contain valued properties. This class/object hybrid concept addresses the issues of using an element as object and class at the same time.

Within the standard ISO 24744 [ISO 2007], these two modelling patterns are combined to construct a method from the meta-model by making the object facet of the clabject an instance of the powertype class, and the class facet of the clabject a subclass of the partitioned type. For example, a method engineer can define, in the method level, the clabject requirements

specification document as an instance of DocumentKind and as a subclass of Document. By using clabjects at the method level, every single element susceptible of being instantiated during enactment is represented by a class, which is appropriate for instantiation, and by an object, which is appropriate for automated manipulation by tools [ISO 2007].

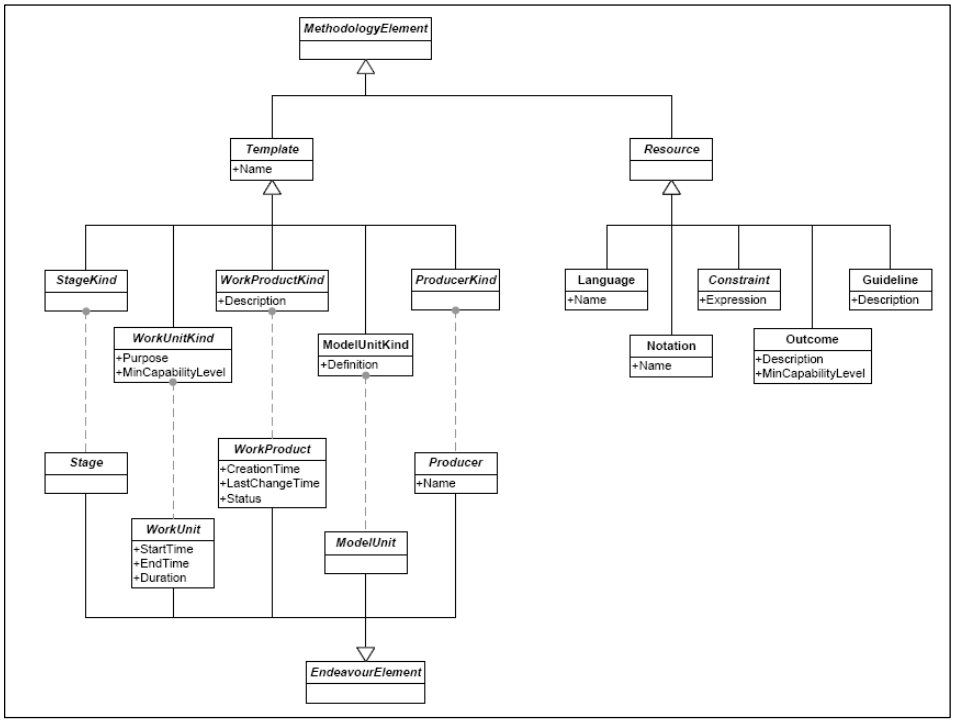

Figure A.16. ISO 24744 Meta-Model [ISO 2007]

As illustrated by Figure A.16, the meta-model ISO 24744 is divided into two groups of core classes:

- A *Resource* is a method element that is directly used at the endeavour level, without an instantiation process. Indeed, any method element that serves as a reference or guideline during an endeavour is represented by Resource. This class is specialized into:
  - *Language* which represents a structure of model unit kinds that focuses on a particular modelling perspective.
  - *Notation* which represents a concrete syntax, usually graphical, which can be used to depict models created with certain languages.
  - *Guideline* which represents an indication of how some method elements can be used.
  - *Constraint* which represents a condition that holds or must hold at a certain point in time.
  - *Outcome* which represents an observable result of the successful performance of a work unit.
- A *Template* is a method element that is used at the endeavour level through an instantiation process. Any method element that acts as a class to be instantiated during enactment as an endeavour element is represented by Template. The powertype pattern formed by Template is refined into more specialized powertype patterns formed by subclasses of these two, namely: WorkUnitKind

and WorkUnit (the element that is performed by producers when developing work products), WorkProductKind and WorkProduct (the element that is developed during an endeavour of a method), and ProducerKind and Producer (the element that produces a work product).

In the following we will focus on details of the ISO meta-model template element.

A5.1. Producer Kind and Producer

*Producer/*Kind* represents an agent that has the responsibility to execute work units. Producers are usually people or groups of people, but can also be tools. As depicted in Figure A.17, the *Producer/*Kind* meta-model defines several important elements like:

- *A role/*Kind* is a collection of responsibilities that a producer can take. *Roles* are often used to declare what responsibilities must be addressed without deciding on how they will be implemented.
- *A team/*Kind* is an organized set of producers that collectively focus on common work units.
- *A tool/*Kind* is an instrument that helps another producer to execute its responsibilities in an automated way.
- *A person/*Kind* is an individual human being involved in a development effort.

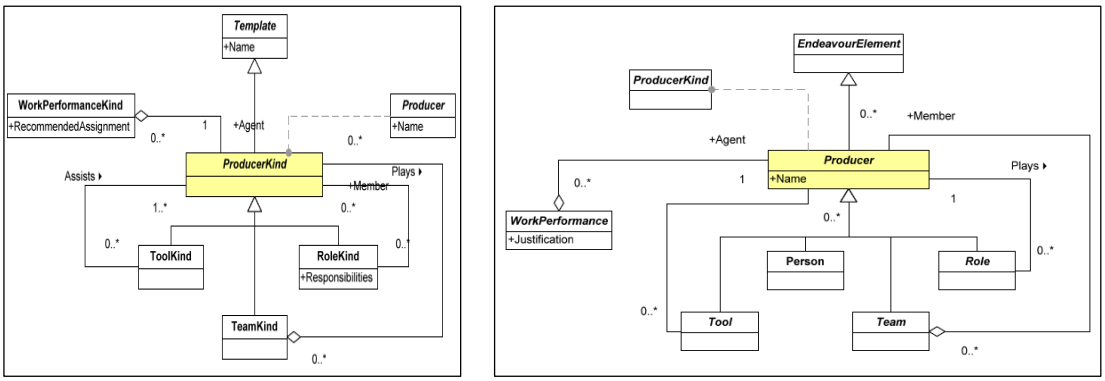

Figure A.17. ISO 24744 Producer/*Kind Meta-Model package [ISO 2007]

A5.2. Work Unit Kind and Work Unit

The *WorkUnit/*Kind* is a job performed, or intended to be performed, within an endeavour. Indeed, a work unit is characterized by a start time (the point in time at which the work unit is started), an end time (the point in time at which the work unit is finished) and a duration (the span of time between the start time and the end time). As depicted in Figure A.18, the *WorkUnit/*Kind* meta-model defines several important elements like:

- A *task/*Kind* is a small-grained work unit that focuses on what must be done in order to achieve a given purpose.
- A *technique/*Kind* is a small-grained work unit that focuses on how the given purpose may be achieved.
- A *process/*Kind* is a large-grained work unit that operates within a given area of expertise.

- A *work performance/*Kind* is an assignment and responsibility association between a particular producer and a particular work unit.

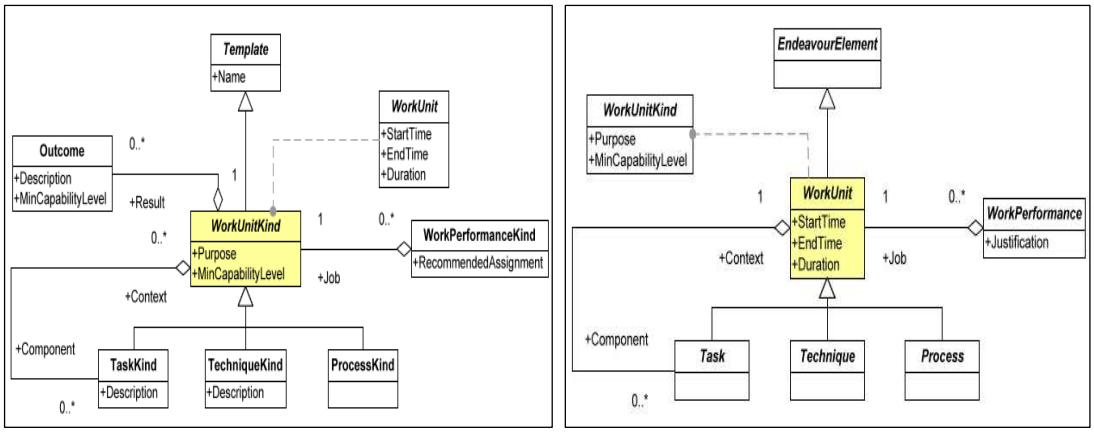

Figure A.18. ISO 24744 Work Unit/*Kind Meta-Model package [ISO 2007]

A5.3. Work Product Kind and Work Product

A *Work Product/*Kind* is an artefact of interest for the endeavour. Work products can be documents, physical things or information collections that are created, or modified during the endeavour. As depicted in Figure A.19, the *Work Product/*Kind* meta-model defines several important elements like:

- A *Composite Work Product/*Kind* is a work product composed of other work products.
- An *Action/*Kind* is a usage event performed by a task upon a work product. Actions represent the fact that specific tasks use specific work products.
- A *Document/*Kind* is a durable depiction of a fragment of reality. Documents often represent models, but they can also represent other subjects.
- A *Software/*Kind* item is a piece of software of interest to the endeavour.

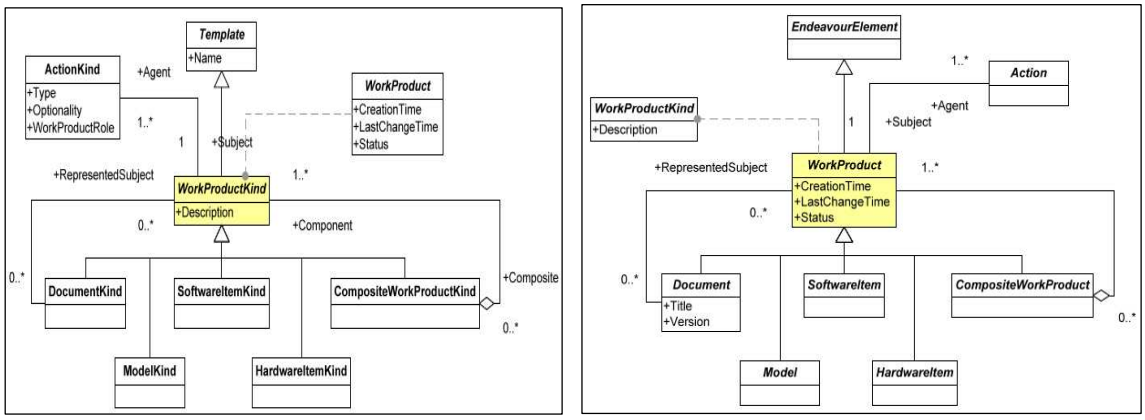

Figure A.19. ISO 24744 Product/*Kind Meta-Model package [ISO 2007]

Note that the ISO 24744 standard defines also additional method elements like (see Figure A16 and Figure A.19):

- *A Stage/*Kind* which represents a managed time frame within an endeavour.
- A *Model Unit/*Kind* which represents an atomic component of a model. Model units are usually linked to each other to form the semantic network that comprises the model. Furthermore, each model unit can appear in multiple models, thus achieving model connectivity.

The ISO 24744 standard uses different diagrams that help to describe the elements of a method according to different perspectives. For example, the *Lifecycle Diagram* represents the overall structure of a method specified with stage elements and work unit elements. The Figure A.20 shows the ISO 24744 lifecycle diagram of the User Interface development method. This diagram presents the stage of the method by using a pentagon symbol (Task and domain construction phase, transformation phase, and generation phase). The activities of each stage are defined within the pentagon. If the activities can be performed in parallel, then a hexagon symbol is used. Finally, a diamond symbol is used to represent the event of the end of a stage. This event allows launching the next stage.

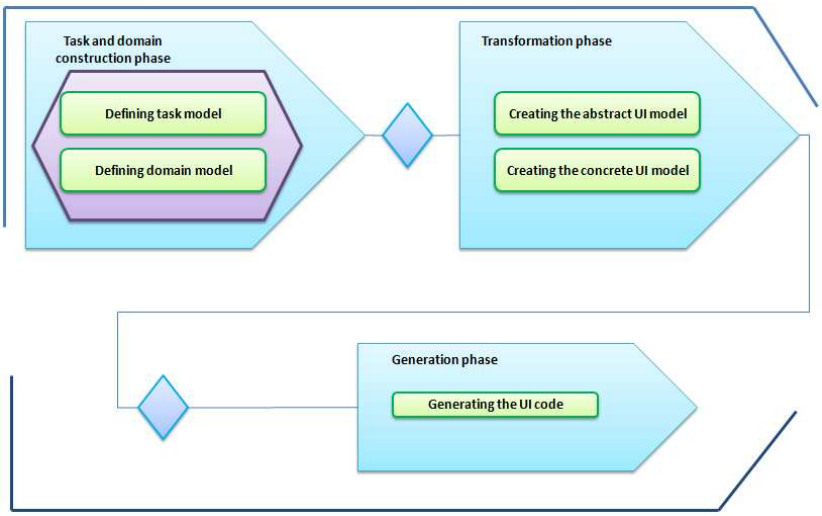

Figure A.20. The ISO 24744 lifecycle diagram of the User Interface development method

Another example of the ISO 24744 standard diagrams is the *Process Diagram* that describes the details of the processes used in a method. It depicts the relationships between work product kind, work unit kind, and producer kind. The Figure A.21 shows the ISO 24744 process diagram of the User Interface development method. In this diagram, a work product is represented by an oval symbol. In turn, a work unit is represented by a process flowchart symbol; finally, a producer is represented by a delay flowchart symbol.

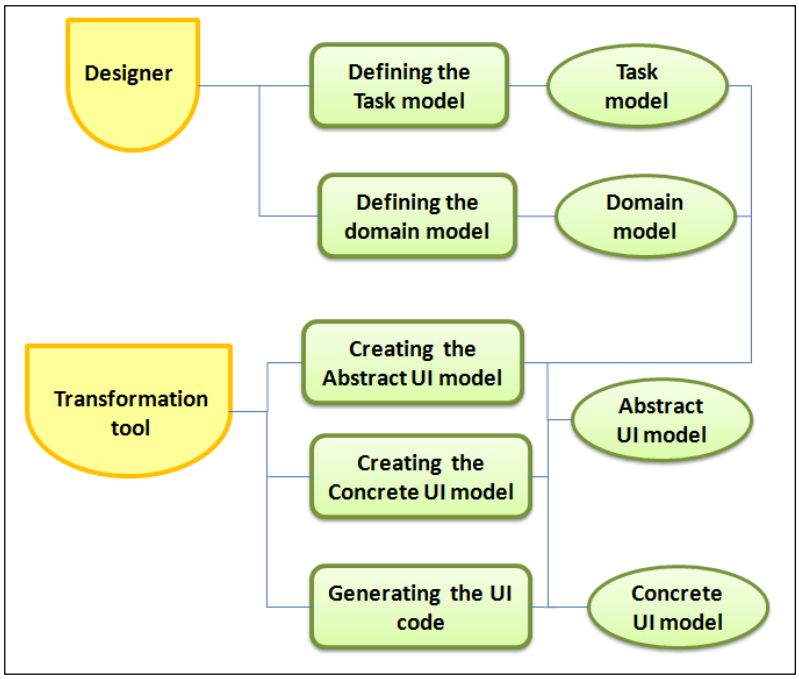

Figure A.21. ISO 24744 Process Diagram of the User Interface development method

A6. COMPARISON BETWEEN THE METHOD META-MODELS

In the previous sections, we presented three meta-model standards for method development. Each of them has its key features, and describes the core elements of a method in a specific way. In this section, we compare these method meta-models according to their key features and their support to method concepts.

The table 1 represents a synthesis of the key features and the limitations of the three meta-model standards for method development.

	Key features	Limitations
SPEM [OMG 2008]	1- SPEM 2.0 is a UML profile. This means that SPEM reuses UML diagrams to describe the elements of a method, which provides a great usability of this standard. 2- SPEM 2.0 separates the operational aspect of a method (*Method Content*) from the temporal aspect of a method (*Process Structure*). This allows using any modelling language to describe the process behaviour. 3- SPEM 2.0 contains abstract	1- SPEM does not allow the method engineer having a control on the endeavour layer from the meta-model layer. The method engineer can define what elements will exist in the method layer, but characterizing endeavour layer elements is not possible [Hen 2005]. 2- SPEM proposes several generic and abstract classes. This leads to have a complex meta-model that can be difficult to understand.

	generalization classes (e.g. Kind element, see Section A3.1) for refinement of the vocabulary used to describe concepts or the relationships between concepts. Another advantage of these abstract generalization classes is the fact that they allow creating a customable method meta-model (UML stereotype mechanism) specific to a certain domain (e.g. User Interface Development). The customized meta-models can be associated with OCL constraints in order to ensure the methods coherence.	
OPEN [OPF 2005]	OPEN provides significant details to describe the different elements of a method.	1- Like SPEM, the OPEN standard does not allow having control on the endeavour layer from the meta-model layer [Hen 2005]. 2- The OPEN standard does not support the abstract generalization classes that allow describing how to create customizable method meta-models.
ISO 24744 [ISO 2007]	The ISO 24744 standard uses a dual-layer modelling to allow the method engineer to configure the enactment of the method from the meta-model level.	Object-oriented programming languages (like JAVA) do not support the dual-layer modelling [Kuh 2007, Gut 2008]. This is an issue since, in the UsiXML project, we plan to use Java based platforms (e.g. Eclipse, GMF, EMF, etc.) in order to develop the UsiXML support tools [Def 2010].

The table 2 represents a synthesis of the support to method concepts in the three meta-model standards for method development.

Method concepts		SPEM 2.0 [OMG 2008]	OPEN [OPF 2005]	ISO 24744 [ISO 2007]
Work Unit (WU)	*WU definition*	*WorkDefinition, TaskDefinition, Step, Task Use*	Work Unit	Work Unit
	WU type	*Activity, Milestone*	Activity, Task, Work Flow, Technique	Process, Task, Technique
	Time unit	*WorkBreakdownElement*	Stage	Stage
	Time unit type	*Kind (Core)*	Phase, Build, Cycle, Milestone, Inch Pebble	Phase, Life-Cycle

Work Product (WP)	*WP definition*	*WorkProductUse, WorkProductDefinition*	Work Product	Work Product
	WP type	*Kind (Core)*	Document, diagram, model, class, application, …	Composite Work Product, Document, Model, Software, …
Producer	*Producer definition*	*WorkDefinitionPerformer, ProcessPerformer, RoleDefinition*	Producer	Producer
	Producer type	*Kind (Core)*	Organization, team, role, person, tool	Team, role, tool, person
Relationships between concepts	*Relationships between WU and WP*	*Default task definition parameter, Process parameter*	Represented as a “manipulate” association relationship between the two concepts	Action
	Relationships between WU and Producer	*Default task definition performer, Process performer*	Represented as a “Perform” association relationship between the two concepts	Work performance
	Relationships between Producer and WP	*Default responsibility assignment, process responsibility assignment*	Represented as a “Produce” association relationship between the two concepts	--
Additional elements	*Constraint*	*Constraint, Precondition, Post condition*	Constraint, Precondition, Goal	Constraint, Precondition, Post condition
	Relationships between Work products	*Work products definition relationship*	--	--

A7. APPENDIX REFERENCES

[ArcStyler 2010] http://www.interactive-objects.com/

[Atk 2000] C. Atkinson, T. Kuhne, “Meta-level independent modeling”. In International Workshop on Model Engineering at 14th European Conference on Object-Oriented Programming, Sophia Antipolis and Cannes, France, 2000.

[Def 2010] Defimedia, 2010, "UsiXML Software tools requirements specification". In the deliverable of the Work package 3: Task 3.1, September 2010

[EPF 2010] Eclipse Foundation, 2010, "Eclipse Process Framework version 1.5.1", http://www.eclipse.org/epf/

[eTrack 2010] http://www.etrack.com.au/

[Gut 2008] M. Gutheil, B. Kennel, C. Atkinson, 2008, "A Systematic Approach to Connectors in a Multi-level Modeling Environment". Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems (MoDELS 2008): 843-857

[Hen 2005] B. Henderson-Sellers and C. Gonzalez-Perez, 2005, "A comparison of four process metamodels and the creation of a new gen@eric standard". Information and Software Technology. Volume 47, Issue 1, 1 January 2005, Pages 49-65

[Hen 2008] B. Henderson-Sellers and C. Gonzalez-Perez, 2005, "Metamodelling for Software Engineering". ISBN-13: 978-0470030363Wiley (October 14, 2008)

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

[Kuh 007] T. Kuhne; D. Schreiber; 2007, "Can Programming be Liberated from the Two-Level Style? : Multi-Level Programming with DeepJava" in OOPSLA'07 International Conference on Object-Oriented Programming, Systems, Languages, & Applications No22, Montréal , CANADA (21/10/2007)

[Lim 2009] Q. Limbourg, J. Vanderdonckt, 2009. "Multipath Transformational Development of User Interfaces with Graph Transformations". In Human-Centered Software Engineering, Human-Computer Interaction Series, Volume . ISBN 978-1-84800-906-6. Springer London, 2009, p. 107

[Myriad 2010] http://myriadinc.net/

[NoMagic 2010] No Magic, 2010, "MAGICDRAW", https://secure.nomagic.com/spem_plugin

[Ode 1994] J.J. Odell, 1994, "Power types". In the Journal of Object-Oriented Programming 7 (2), (1994) 8–12.

[OMG 2002] OMG, 2002 "Software & Systems Process Engineering Meta-Model Specification version 1.0", In OMG Document Number: formal/02-11-14. Standard document URL: http://www.omg.org/cgi-bin/doc?formal/02-11-14.pdf

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

[OPEN 2010] OPEN Consortium, 2010, "OEPN", http://www.open.org.au/

[OpenUP 2010] http://epf.eclipse.org/wikis/openup/

[OPF 2005] OPEN Process Framwork, 2005, "OPF Meta-Model", http://www.opfro.org/index.html?Overview/Metamodel.html~Contents

[Scrum 2010] http://www.scrum.org

[XP 2010] http://www.extremeprogramming.org/