Deliverable D2.4.1 - Collaborative MDE Process Modeling

Task T.2.4 - Collaborative MDE Process Modelling and assisted enactment

	NAME	***PARTNER***	***DATE***
WRITTEN BY	LBATH R. (sections 1, 2) KEDJI E. (section 3) TRAN H. N. (section 3) COULETTE B. (section 4) EBERSOLD S. (section 4)	IRIT	01/01/2011
REVIEWED BY			

RECORD OF REVISIONS

ISSUE	***DATE***	***EFFECT ON***		***REASONS FOR REVISION***
		PAGE	***PARA***	
1.0	December 13th, 2010			Document creation

TABLE OF CONTENTS

TABLE OF APPLICABLE DOCUMENTS

N°	TITLE	REFERENCE	ISSUE	DATE	SOURCE	
					SIGLUM	NAME
A1	?????					
A2	?????					
A3	?????					
A4	?????					

TABLE OF REFERENCED DOCUMENTS

N°	TITLE	REFERENCE	ISSUE
R1	Galaxy glossary		
R2	?????		
R3	?????		
R4	?????		

ACRONYMS AND DEFINITIONS

Except if explicitly stated otherwise the definition of all terms and acronyms provided in [R1] is applicable in this document. If any, additional and/or specific definitions applicable only in this document are listed in the two tables below.

Acronymes

ACRONYM	***DESCRIPTION***

Definitions

TERMS	***DESCRIPTION***

1. INTRODUCTION

1.1 GOAL OF THIS DOCUMENT

The work presented in the present document is a part of the task T2.4 of the Galaxy work package WP2. The objective of this task is to define concepts and a methodology for modeling processes that govern model-driven collaborative development, so called Collaborative MDE Processes. The final aim is to use such process models in order to provide a computer-assisted enactment.

The main issues addressed in task T2.4 are:

1. Process structure. As we know from the existing process modelling formalisms (e.g. SPEM [OMG, 2008-a]), a development process is described in terms of activities, artifacts produced or consumed by activities, roles played by human actors, elements of guidance provided for human actors, and tools involved in development. The issue is to identify how such formalisms have to be extended and/or adapted to collaborative MDE processes.

2. Viewpoint-oriented Process Modelling. To face the scalability issue in the context of complex systems, separation of concerns has proven to be efficient. The issue is to study how the existing viewpoint-oriented modelling approaches and particularly our previous work (e.g. [Marcaillou94], [Nassar05]) could be applied to collaborative MDE development.

3. Process-based assistance. To assist users during the collaborative MDE process enactment, guidance and behavior aspects should be incorporated into the process structure description. The issue is to identify such aspects in order to allow computer-assisted enactment.

4. Collaborative process-based traceability. A particular way to assist users during the process enactment is to provide traces according to users' viewpoints. Traces might be used for several purposes such as guidance, process debugging, and reverse engineering. The issue is to define a traceability policy in the context of collaborative MDE development, and to integrate it into the process of computer-assisted enactment.

5. Flexible Process definition. A well-known problem in enacting software process models is that processes are subject to permanent deviations, due to many reasons. The issue is to study how the existing approaches to process evolution management (e.g. [Kabbaj, 2008], [Almeida, 2010]) could be applied in the context of collaborative MDE development.

The present deliverable D2.4.1 focuses on issues 1, and part of issue 2. The remaining issues will be addressed in the deliverable D2.4.2 (Collaborative MDE Process Assistance Definition). In other words, D2.4.1 focuses only on structural and static aspects of collaborative MDE processes, while dynamic aspects (related to process enactment) and methodological aspects will be presented in D2.4.2. Part of issue 2 (methodological aspect) will be treated in D2.4.2 since it is related to process designer assistance.

The present document aims to depict the Galaxy's conceptual approach for collaborative MDE process structure modeling. By MDE process structure, we mean entities that characterize MDE processes, such as activities, artifacts, roles, models, transformations, etc., and their relationships. The approach consists in providing a well-defined language in the form of a model, called CM_SPEM (Collaborative Model-based Software & Systems Process Engineering Metamodel), that extends the OMG's standards SPEM 2.0 [OMG, 2008-a] and QVT [OMG, 2008-b] to concepts related to collaborative MDE development.

1.2 DOCUMENT ORGANIZATION

The remainder of this document is organized into three main sections: section 2 deals with modeling MDE process structure aspects independent of the collaborative and the view-points concerns; section 3 focuses on modeling structural aspects of collaborative processes, including MDE processes. Section 4 deals with structural concepts that are related to viewpoint process modeling.

2. MDE PROCESS STRUCTRE MODELLING

2.1 INTRODUCTION

MDE development is usually described as an Engineering approach that promotes the use of models and transformations as primary artifacts throughout the development process. In MDE development, the main preoccupation of the developers is to design models that capture the various concepts and relations the system to be built is made of, and to identify model transformations that lead to the effective construction of the system. So, the finality of MDE development is to describe both the problem and its solution by using models, and by clearly establishing a methodology to show how to switch from a problem, described by the model of requirements of the system to be built, to its solution, i.e. the effective construction of the system, by using models transformations. Therefore, a MDE process can be seen as a set of model transformations, each transformation consuming source models and producing target models. To allow computer-based transformations, models should not be informal. They must conform to precise metamodels or be written in a well-defined language, i.e. a language with a precise syntax and semantics, that make them machine understandable. Thus, a certain number of transformations, such as model refactoring, model refinement, or model to code generation, may be at least partially automated.

The SPEM 2.0 standard describes a process in terms of process activities, work products consumed and/or produced by activities, roles played by human actors that perform activities, and guidance elements and tools providing assistance for human actors at enactment time. However, SPEM 2.0 does not explicitly offer concepts related to MDE development. In the other hand, the QVT standard provides concepts and constructs for expressing model transformations, but it does not address the question of process modeling.

The basic approach underlying the definition of CM_SPEM Process Structure metamodel consists in reusing a subset of SPEM 2.0 and QVT in order to include concepts related to MDE processes. A model is defined as a specialization of a SPEM work product. Thus, activities may work on models.

The concept of model transformation is defined as a specialization of SPEM work definition that has models as input/output parameters. It is also defined as a SPEM work breakdown element. Thus, model transformations may be nested by SPEM activities.

The concept of Process is considered as a kind of Activity. Similarly, in CM_SPEM, the concept of MDE process is considered as a kind of Activity, which may contain activities that work on models, including model transformation, model edition, model refactoring, etc.

2.2 CM_SPEM PROCESS STRUCTURE OVERVIEW

As depicted in **Erreur ! Source du renvoi introuvable.**, the CM_SPEM Process Structure metamodel merges the packages SPEM 2.0 Core, SPEM 2.0 Process Sructure, and QVT Base, and is organized into five sub-packages:

- CM_SPEM Core
- CM_SPEM Model Structure
- CM_SPEM Transformation Structure
- CM_SPEM Activity Structure

The SPEM 2.0 Core package contains the meta-model classes and abstractions that build the foundation for structural aspects of processes. It is reused by CM_SPEM Core via the UML 2 package merging mechanism.

The CM_SPEM Model Structure package defines concepts of models and metamodels and their relationships. It imports concepts from the UML2 Core and SPEM 2.0 Core packages.

The SPEM 2.0 Process Structure package defines the base for all process models. Its core data structure is a breakdown or decomposition of nested Activities that maintain lists of references to performing Role classes as well as input and output Work Product classes for each Activity. In addition, it provides mechanisms for process reuse such as the dynamic binding of process patterns that allow users to assemble processes with sets of dynamically linked Activities.

The QVT Base package contains a set of basic concepts that structure transformations, their rules, and their input and output models. The packages QVT Base, SPEM 2.0 Process Structure, CM_SPEM Core, and CM_SPEM Model Structure are merged into the CM_SPEM Transformation Structure package in order to extend SPEM work definitions to model transformations' structural aspects.

The packages SPEM 2.0 Process Structure, CM_SPEM Core, CM_SPEM Model Structure, CM_SPEM Transformation Structure are merged into the CM_SPEM Activity Structure package in order to extend activities' structural aspects with the concepts of model and model transformation.

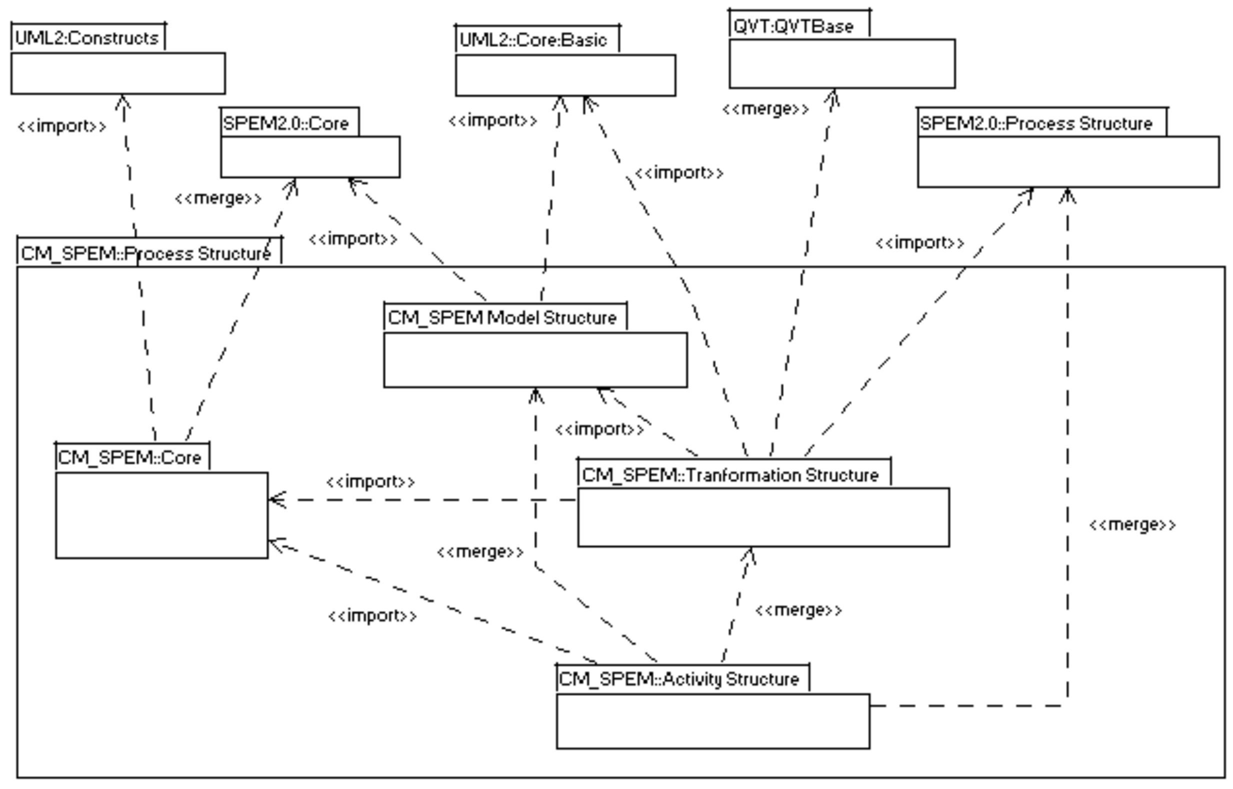

Figure 1 - CM_SPEM Process Structure: sub-packages

2.3 CM_SPEM CORE

This package contains the meta-model classes and abstractions that build the foundation for all other meta-model packages. It imports the concepts Classifier, and Class from UML2, and merges the SPEM2.0 Core package.

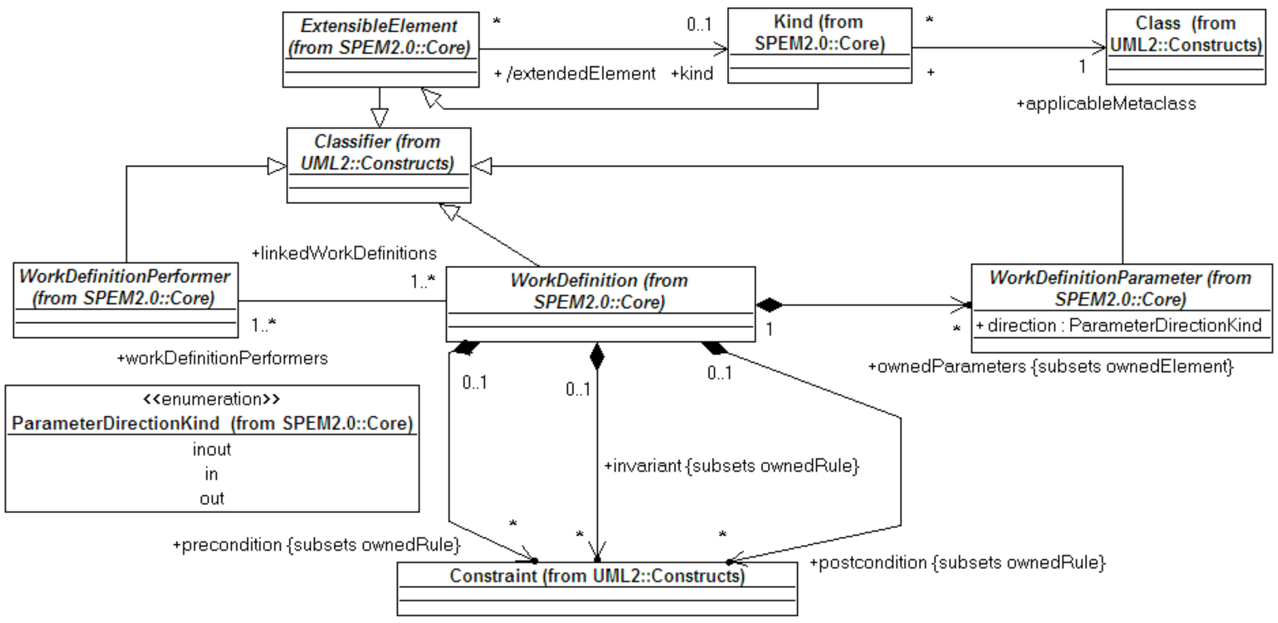
ExtensibleElement (from SPEM2.0::Core)
Kind (from SPEM2.0::Core)
Class (from UML2::Constructs)
+ /extendedElement
+kind
+applicableMetaclass
Classifier (from UML2::Constructs)
+linkedWorkDefinitions
WorkDefinitionPerformer (from SPEM2.0::Core)
WorkDefinition (from SPEM2.0::Core)
WorkDefinitionParameter (from SPEM2.0::Core)
+ direction : ParameterDirectionKind
+workDefinitionPerformers
+ownedParameters {subsets ownedElement}
<<enumeration>> ParameterDirectionKind (from SPEM2.0::Core)
inout
in
out
+invariant {subsets ownedRule}
+precondition {subsets ownedRule}
+postcondition {subsets ownedRule}
Constraint (from UML2::Constructs)

Figure 2 - The CM_SPEM Core package

2.3.1 Newly Introduced Concepts

The CM_SPEM Core package extends the SPEM2.0 Core package with the concept of ‘invariant’ associated with a Work Definition, which is defined as a logical set of constraints that must hold while the Work Definition is being performed.

2.3.2 Extensible Element

Super Class

- Classifier (from Constructs in UML 2 Infrastructure)

Description

Extensible Element is an abstract generalization that represents any class for which it is possible to assign a Kind to its instances expressing a user-defined qualification.

Association Properties

- kind: Kind. An instance of Extensible Element can be linked to zero or one Kind in which the Kind instance expresses a specific user-defined qualification for that Extensible Element instance.

Semantics

Extensible Element provides the property of relating a Kind class to its sub-classes. Such Kinds cannot be reused for many different subtypes of Extensible Element and therefore can only be related to exactly one meta-model class. This is defined as an OCL constraint in SPEM2.0, named “Applicable MetaClass”.

2.3.3 Kind

Super Class

- Extensible Element (from SPEM2.0 Core)

Description

Kind is an Extensible Element. It instances are used to qualify other Extensible Element instances with a user-defined type or kind.

Association Properties

- applicableMetaClass: Class. An instance of Kind can only be used for instances of exactly one Extensible Element subclass or its subclasses. This property specifies which one.

Semantics

As many processes need to define their own refined vocabulary, Kind provides the ability to express such user-defined qualifications for instances of Extensible Element. Because Kind is an Extensible Element itself one can define Kinds for the Kind class itself as well. For example, a subclass of Extensible Elements that typically utilizes Kinds is the meta-model class Guidance. Typical Guidance kinds would be: White Paper, Guideline, Checklist, Template, Reports, etc. Because of the Applicable MetaClass constraint, these Kinds can only be related to instances of the Guidance class as well as instances of any subclasses of Guidance.

2.3.4 ParameterDirectionKind

Description

This enumeration defines for Work Definition Parameter instances whether the parameter represents an input, output, or input as well as output.

Enumeration Literals

- in: A Work Definition Parameter instance with this direction value represents an input.
- out: A Work Definition Parameter instance with this direction value represents an output.
- inout: A Work Definition Parameter instance with this direction value represents both an input and an output.

2.3.5 WorkDefinition

Super Class

- Classifier (from Constructs in UML 2 Infrastructure)

Description

Work Definition is an abstract Classifier that generalizes all definitions of work. Work Definition defines some default associations to Work Definition Parameter and Constraint. Work Definitions can contain sets of pre-conditions, invariants, and post-conditions defining constraints that, respectively, need to be valid before the described work can begin, while it is being performed, or before it can be declared as finished.

Association Properties

- /ownedParameters: WorkDefinitionParameter. Work Definition can define an ordered set of parameters to specify inputs and outputs. The concrete subclasses of Work Definition need to define their own subclasses of Work Definition Parameter to add reference to concrete input/output meta types.
- workDefinitionPerformers: WorkDefinitionPerformer. This composition association specifies the performers of the work described by the Work Definition.
- precondition: Constraint. This composition association adds an optional pre-condition to a Work Definition. A pre-condition defines any kind of constraint that must evaluate to true before the work described by the Work Definition can start.

- postcondition: Constraint. This composition association adds an optional post-condition to a Work Definition. A post-condition defines any kind of constraint that must evaluate to true before the work described by the Work Definition can be declared completed or finished and which other Work Definitions might depend upon (e.g., for their pre-conditions).
- Invariant: Constraint. This composition association adds an optional invariant to a Work Definition. An invariant defines any kind of constraint that must hold before the work described by the Work Definition can start, while it is being performed, and before it can be declared completed or finished.

Semantics

A Work Definition represents a performer independent definition of work. For example, a Work Definition could represent work that is being performed by one specific Role (e.g., a Role performing a specific Activity), by many Roles working in close collaboration (many Roles all working together on the same Activity), or complex work that is performed throughout the lifecycle (e.g., a process defining a breakdown structure for organizing larger composite units of work performed by many Roles working in collaboration).

2.3.6 WorkDefinitionParameter

Super Class

- Classifier (from Constructs in UML 2 Infrastructure)

Description

A Work Definition Parameter is an abstract generalization for Process Elements that represent parameter for Work Definitions. It is used for declarations of inputs and outputs.

Attributes

- direction: ParameterDirectionKind. This attribute represents the direction kind of the parameter as specified by the enumeration Parameter Direction Kind.

2.3.7 WorkDefinitionPerformer

Super Class

- Classifier (from Constructs in UML 2 Infrastructure)

Description

Work Definition Performer is an abstract Classifier that represents the relationship of a work performer to a Work Definition. Different specialization of Work Definition will introduce different kinds of performers. Work Definition Performer is intended to be specialized adding the association to the concrete performer meta class.

Association Properties

- /linkedWorkDefinition: WorkDefinition This derived union provides access to all the Work Definitions a Work Definition Performer instance is related to.

2.4 CM_SPEM MODEL STRUCTURE

This package defines concepts of Model, Metamodel, and their main related relationships. It imports the concept of Named Element from UML2 and the concept of Extensible Element from the SPE2.0 Core package.

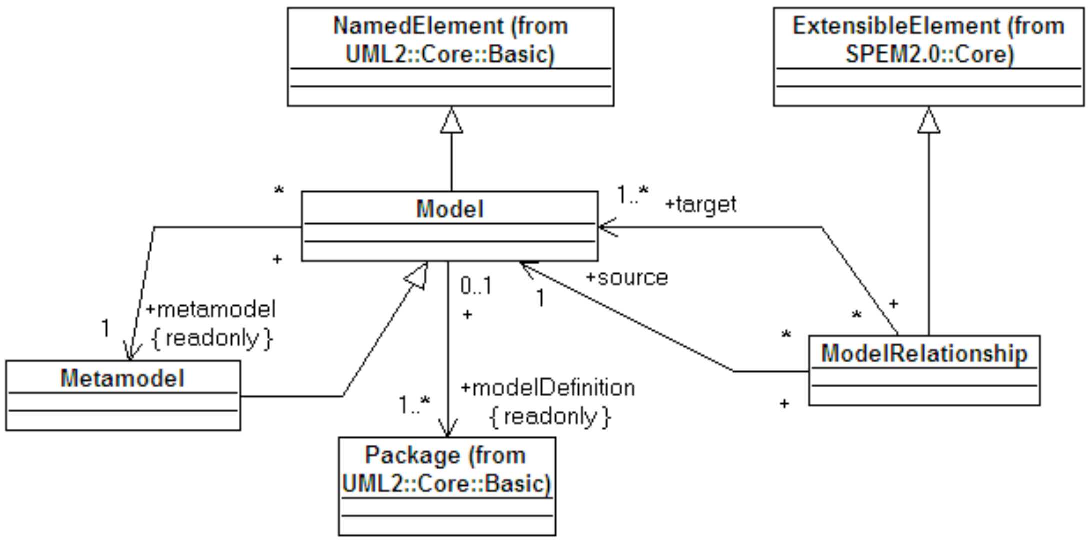
NamedElement (from UML2::Core::Basic)
ExtensibleElement (from SPEM2.0::Core)
Model
Metamodel
Package (from UML2::Core::Basic)
ModelRelationship
*
+
1
+metamodel {readonly}
0..1
+
1..*
+modelDefinition {readonly}
1..*
+target
+source
1
*
+
*
+

Figure 3 - The CM_SPEM Model Structure package

2.4.1 Model

Super Class

- Named Element (from UML2::Core::Basic)

Description

Model defines a description of a System that conforms to a metamodel. It comes in the form of a non-empty set of packages that contains elements defining the model.

Association Properties

- metamodel: Metamodel. The Metamodel that the Model conforms to.
- modelDefinition: Package (from UML2::Core:Basic). The non-empty set of packages that contains elements defining the model.

2.4.2 Metamodel

Description

Metamodel is a particular model that defines an abstract language for describing a given family of models. As a Model, it must conform to a Metamodel, itself eventually, and must have a non-empty set of packages that contains elements defining it.

2.4.3 ModelRelationship

Super Class

- Extensible Element (from SPEM2.0 Core)

Description

Model Relationship expresses a general relationship among models. Kind class (from Core) instances shall be used to specify the nature of this relationship.

Association Properties

- source: Model. This association links to the exact one source of the Model Relationship.
- target: Model. This association links to one or more targets of the Model Relationship.

Semantics

Model relationships can be used to express model-specific kinds of relationships among Models, such as refinement, traceability, etc.

2.5 CM_SPEM TRANSFORMATION STRUCTURE

This package defines the concept of Transformation as a Work Definition that has Typed Models as parameters, and Process Performers. It imports the following concepts: Package from UML2, Work Definition from CM_SPEM Core, Process Performer and Work Definition Parameter from SPEM2.0 Process Structure. It merges the QVT::QVT Base package and the CM_SPEM Model Structure package.

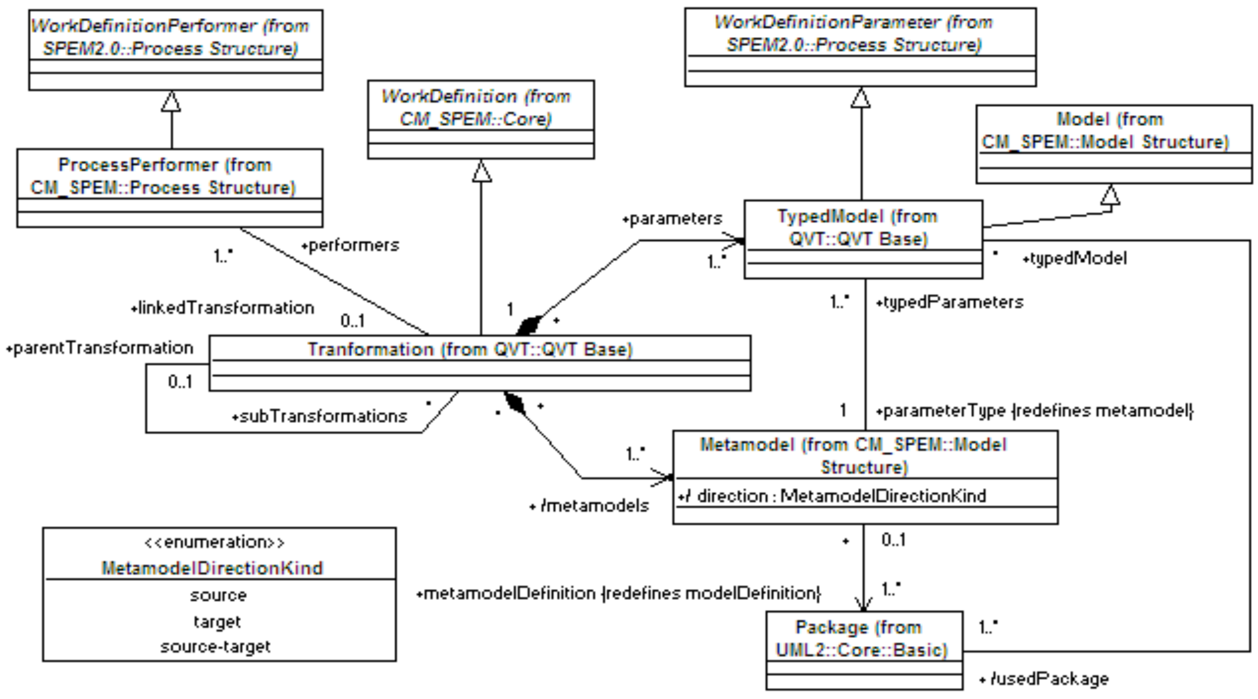
WorkDefinitionPerformer (from SPEM2.0::Process Structure)
WorkDefinitionParameter (from SPEM2.0::Process Structure)
WorkDefinition (from CM_SPEM::Core)
Model (from CM_SPEM::Model Structure)
ProcessPerformer (from CM_SPEM::Process Structure)
TypedModel (from QVT::QVT Base)
+parameters
+performers
+typedModel
+linkedTransformation
+typedParameters
+parentTransformation
Tranformation (from QVT::QVT Base)
+subTransformations
+parameterType {redefines metamodel}
Metamodel (from CM_SPEM::Model Structure)
+/ direction : MetamodelDirectionKind
+ /metamodels
<<enumeration>>
MetamodelDirectionKind
source
target
source-target
+metamodelDefinition {redefines modelDefinition}
Package (from UML2::Core::Basic)
+ /usedPackage
1..*
0..1
1
0..1
*

Figure 4 - CM_SPEM Transformation Structure package

2.5.1 Newly Introduced Concepts

The CM_SPEM Transformation Structure package extends the QVT::QVT Base package with the following concepts:

- As a Work Definition, a QVT Transformation may have a precondition, an invariant, and a post-condition.
- As a Work Definition, a QVT Transformation has Process Performers.
- A QVT Transformation is not only associated with models as parameters but also with the metamodels that these models must conform to.
- Relationships between Typed Models, which are parameters of a QVT Transformation, may be explicitly defined.

2.5.2 Transformation

Super Class

- WorkDefinition (from CM_SPEM Core)
- WorkBreakdownElement (from CM_SPEM Activity Structure)

Description

Transformation defines how one set of models can be transformed into another. Types of models are specified by a set of typed model parameters associated with the transformation. It contains a set of rules that specify how models are to be transformed.

As a Work Definition, a Transformation represents a piece of work within a model-based development process. It has Work Definition Parameters (which correspond to Typed Models), Work Definition Performers, and may have precondition, invariant, and postcondition.

As a Work Breakdown Element, a Transformation may be nested by Activties of a model-based development process, and may have Work Sequence links with other Work Breakdown Elements.

Association properties

- performers: ProcessPerformer (from SPEM2.0::Process Structure). This association specifies the performers of the work described by the Transformation. It subsets the workDefinitionPerformers association inherited from Work Definition.
- parameters: TypedModel (from QVT::QVT Base). This composition association specifies an

ordered set of typed model parameters for the Transformation.

• /metamodels: Metamodel (from CM_SPEM Model Structure). This composition association specifies the set of Metamodels used by the Transformation, which is the union of Metamodels that type the Transformation's parameters.

• subTransformations: Transformation. This association may be used to decompose the Transformation into a set of sub-Transformations.

• parentTransformation: Transformation. This is the opposite of the association subTransformations. It indicates the parent Transformation, if any.

2.5.3 TypedModel

Super Class

- Work Definition Parameter (from SPEM2.0::Process Structure)
- Model (from CM_SPEM Model Structure)

Description

Typed Model specifies a named, typed parameter of a transformation, which is a model that conforms to a Metamodel.

Association properties

- parameterType: Metamodel (from CM_SPEM Model Structure) {redefines metamodel (from CM_SPEM Model Structure)}. This association specifies Metamodel that the TypedModel conforms to.

usedPackage: Package (from UML2::Core::Basic) {redefines usedPackage (from QVT::QVT Base)}. The non-empty set of metamodel packages that specify the types for the model elements of the Typed Model.

2.5.4 Metamodel

Description

Metamodel specifies the type of one or several model parameters of a Transformation.

Association Properties

- metamodelDefinition: Package (from UML2::Core::Basic) {redefines modelDefinition (from CM_SPEM Model Structure). This association specifies thee non-empty set of packages that contains elements that define the metamodel.
- typedParameters: TypedModel. This association specifies the Typed Models that conform to the Metamodel.

Attributes

- /direction: MetamodelDirectionKind. This attribute represents the direction kind of the parameter as specified by the enumeration Parameter Direction Kind. The value of this attribute is derived from the value of the attribute direction of Models associated with the Metamodel through the association property 'typed parameters'. If all these models have their attribute 'direction' set to 'in', then the attribute 'direction' of the Metamodel is set to 'source'. If all the models their attribute 'direction' set to 'out', then the attribute 'direction' of the Metamodel is set to 'target'. Otherwise, the attribute 'direction' of the Metamodel is set to 'source-target'.

2.5.5 Metamodel Direction Kind

Description

This enumeration defines for a Metamodel used by a Transformation whether it used as a source metamodel, a target metamodel, or both.

Enumeration Literals

- source: A Metamodel with this direction value indicates that it is used as a source metamodel.
- out: A Metamodel with this direction value indicates that it is used as a target metamodel.
- inout: A Metamodel with this direction value indicates that it is used both as a source metamodel and target metamodel.

2.6 CM_SPEM ACTIVITY STRUCTURE

As it is the case in SPEM2.0, this package contains the basic structural elements for defining processes in terms of development activities. Its core data structure is a breakdown or decomposition of nested activities that maintain, for each activity, references to breakdown elements such as performing roles, input and output work products, and milestones. This breakdown structure mechanism is defined independent of the concrete lifecycle models the process engineer wants to express with them. In other words, the meta-model is able to represent different types of processes, such as waterfall processes as well as iterative or incremental process models, by modeling them all as breakdown structures, but applying different structural relationships and descriptive attributes expressing their lifecycle specifics.

The CM_SPEM Activity Structure package imports the CM_SPEM Core package and merges the following packages: SPEM2.0 Process Structure, CM_SPEM Model Structure, and Transformation Structure.

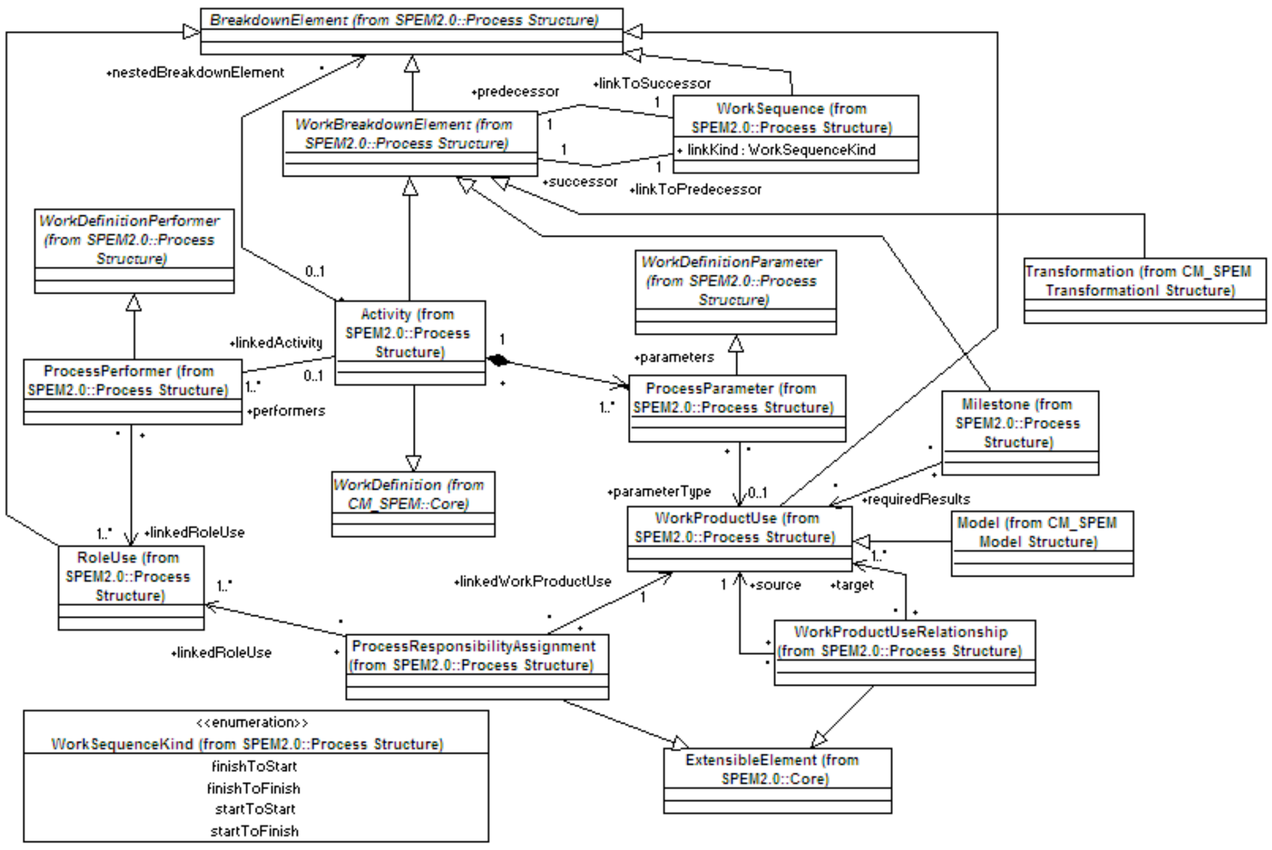
BreakdownElement (from SPEM2.0::Process Structure)
+nestedBreakdownElement
+predecessor
+linkToSuccessor
WorkBreakdownElement (from SPEM2.0::Process Structure)
WorkSequence (from SPEM2.0::Process Structure)
+ linkKind : WorkSequenceKind
+successor
+linkToPredecessor
WorkDefinitionPerformer (from SPEM2.0::Process Structure)
WorkDefinitionParameter (from SPEM2.0::Process Structure)
Transformation (from CM_SPEM TransformationI Structure)
Activity (from SPEM2.0::Process Structure)
+linkedActivity
ProcessPerformer (from SPEM2.0::Process Structure)
+performers
+parameters
ProcessParameter (from SPEM2.0::Process Structure)
Milestone (from SPEM2.0::Process Structure)
WorkDefinition (from CM_SPEM::Core)
+parameterType
+requiredResults
+linkedRoleUse
WorkProductUse (from SPEM2.0::Process Structure)
Model (from CM_SPEM Model Structure)
RoleUse (from SPEM2.0::Process Structure)
+linkedWorkProductUse
+source
+target
ProcessResponsibilityAssignment (from SPEM2.0::Process Structure)
WorkProductUseRelationship (from SPEM2.0::Process Structure)
+linkedRoleUse
<<enumeration>>
WorkSequenceKind (from SPEM2.0::Process Structure)
finishToStart
finishToFinish
startToStart
startToFinish
ExtensibleElement (from SPEM2.0::Core)

Figure 5 - The CM_SPEM Activity Structure package

2.6.1 Newly Introduced Concepts

The CM_SPEM Activity Structure package extends the SPEM2.0 Process Structure, CM_SPEM

Model Structure, and Transformation Structure packages with the following abilities:

- As Work Breakdown Element, QVT Transformations may have precedence relationships (Work Sequences).
- As BreakdownElements, QVT Transformations may be nested by Activities.
- As WorkProductUses, Models may be Process Parameters for Activities.

The concept of MDE Process is considered as special kind of Activity that may contain sub-activities working on Models and QVT Transformations.

2.6.2 Activity

Super Class

- Work Definition (from Core)
- Work Breakdown Element (from SPEM2.0 Process Structure)

Description

Activity is a Work Breakdown Element and Work Definition that defines basic units of work within a process. Activity supports the nesting and logical grouping of related Breakdown Elements forming breakdown structures.

Association Properties

- nestedBreakdownElement: BreakdownElement. This association represents breakdown structure nesting. It defines an n-level hierarchy of Activities grouping together other Breakdown Elements such as other Activities, Milestones, etc.
- parameters: ProcessParameter. This composition association specifies an ordered set of ProcessParameters for the Activity. The association property subsets the ownedParameters association inherited from Work Definition.
- performers: ProcessPerformer (from SPEM2.0::Process Structure). This association specifies the performers of the work described by the Activity. It subsets the workDefinitionPerformers association inherited from Work Definition.

Semantics

Activity is a concrete Work Definition that represents a general unit of work assignable to specific performers represented by Role Use. An Activity can rely on inputs and produce outputs represented by Work Product Uses. Thanks to the association property nestedBreakdownElement, Activity also represents a grouping element for other Breakdown Elements such as Activities (or sub-activities), Milestones, etc.

2.6.3 Breakdown Element

Description

Breakdown Element is an abstract generalization for any element that is part of a breakdown structure. Any of its concrete subclasses can be 'placed inside' an Activity (via the nested Breakdown Element association) to become part of a breakdown of Activities as well as the Activities namespace. As Activities are Breakdown Elements themselves and therefore can be nested inside other activities, an n-level breakdown structure is defined by n nested Activities.

2.6.4 Work Breakdown Element

Super Class

- Breakdown Element (from SPEM2.0 Process Structure)

Description

Work Breakdown Element is a special Breakdown Element that provides specific properties for Breakdown Elements that represent work. The properties are specific to breakdown structures and do not apply to all Work Definition subclasses.

Association Properties

- linkToPredecessor: WorkSequence. This association links a Work Breakdown Element to its predecessor. Every Work Breakdown Element can have predecessor information associated to it. This predecessor information is stored in instances of the class Work Sequence that defines the kind of predecessor another Work Breakdown Element represents for another.
- linkToSuccessor: WorkSequence. This association links a Work Breakdown Element to its successor. Every Work Breakdown Element can have successor information associated to

it. This successor information is stored in instances of the class Work Sequence that defines the kind of successor another Work Breakdown Element represents for another.

Semantics

Work Breakdown Element represents a work-specific breakdown element to be used in a work breakdown structure.

2.6.5 Work Sequence

Super Class

- Breakdown Element (from SPEM2.0 Process Structure)

Description

Work Sequence is a Breakdown Element that 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.

Attributes

- linkKind: WorkSequenceKind. This attribute expresses the type of the Work Sequence relationship by assigning a value from the Work Sequence Kind enumeration.

Association Properties

- successor: WorkBreakdownElement. This association links a Work Breakdown Element to its successor. Every Work Breakdown Element can have successor information associated to it. This successor information is stored in instances of the class Work Sequence that defines the kind of successor another Work Breakdown Element represents for another.
- predecessor: WorkBreakdownElement. This association links a Work Breakdown Element to its predecessor. Every Work Breakdown Element can have predecessor information associated to it. This predecessor information is stored in instances of the class Work Sequence that defines the kind of predecessor another Work Breakdown Element represents for another.

Semantics

The Work Sequence class defines predecessor and successor relations among Work Breakdown Elements. This information is in particular critical for use of the process in planning applications.

2.6.6 Work Sequence Kind

Description

Work Sequence represents a relationship between two Work Breakdown Elements in which one Work Breakdown Element (referred to as (B) below) depends on the start or finish of another Work Breakdown Element (referred to as (A) below) in order to begin or end. This enumeration defines the different kinds of Work Sequence relationships.

Enumeration Literals

- finishToStart Work. Breakdown Element (B) cannot start until Work Breakdown Element (A) finishes.
- finishToFinish. Breakdown Element (B) cannot finish until Work Breakdown Element (A) finishes.
- startToStart. Breakdown Element (B) cannot start until Work Breakdown Element (A) starts.
- startToFinish. Breakdown Element (B) cannot finish until Work Breakdown Element (A) starts.

2.6.7 Work Product Use

Super Class

- Breakdown Element (from SPEM2.0 Process Structure)

Description

Work Product Use is a special Breakdown Element that either represents an input and/or output type for an Activity, or a general participant of the Activity. If it is an input/output, then the Work Product Use needs to be related to the Activity via the Process Parameter class. If it is a participant, then the Work Product Use is stored in the nestedBreakdownElement composition of the Activity and might be used by one of the sub-activities as an input/output and/or be related to a Role Use via a Process Responsibility Assignment. Work Product Use instances are only valid within the context of an Activity and not to be reused across activities.

Semantics

A Work Product Use represents an activity-specific occurrence of a Work Product input/output type or an Activity participant. A Work Product Use instance is an activity-specific object and not a general reusable definition of a work product. A Work Product Use represents the occurrence of a real Work Product in the context of an activity. A Work Product Use participant stored with an Activity can only be accessed and reused by the Activity's sub-Activities and not by any parent or sibling Activities in the Activity breakdown structure. This scoping of Work Product Use in the local namespace of Activities allows the modeling of different Responsibility Assignments for every Activity.

2.6.8 Model

Super Class

- Work Product Use

Description

Model is a special kind of Work Product Use, used as a parameter of an Activity.

Semantics

The use of Models as parameters of Activities allows for definition of works on models that are not model transformations, such as model edition for example. It also allows for making links between Activities and model Transformations, by sharing Models.

2.6.9 Work Product Use Relationship

Super Class

- Extensible Element (from SPEM2.0 Core)

Description

Work Product Use Relationship expresses a general relationship among work products. Kind class (from Core) instances shall be used to specify the nature of this relationship.

Association Properties

- source: Work Product Use. This association links to the exact one source of the Work Product Use Relationship.
- target: Work Product Use. This association links to one or more targets of the Work Product Use Relationship.

Semantics

The Work Product Use Relationship can be used to express different kinds of relationships among Work Products Uses. Typical Kinds are 'composition' expressing that a work product use instance is part of another work product instance, 'aggregation' indicating that a Work Product Use is used with another Work Product Use, and 'impact dependency' indicating that a work product use impacts another work product use.

2.6.10 Process Parameter

Super Class

- Work Definition Parameter (from SPEM2.0::Process Structure)

Description

Process Parameter defines input and output meta-types to be Work Product Uses.

Association Properties

- parameterType: WorkProductUse. This association links zero or one Work Product Use instances to a parameter. Processes could leave the type specification open and not specify a concrete Work Product Use.

2.6.11 Milestone

Super Class

- Work Breakdown Element (from SPEM2.0 Process Structure)

Description

Milestone is a Work Breakdown Element that represents a significant event for a development project.

Association Properties

- requiredResults: WorkProductUse. This association links the Work Product Uses instances to a Milestone instance that need to be produced for that Milestone.

Semantics

A 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). Because Milestone is commonly used to refer to both the event itself and the point in time at which the event is scheduled to happen, it is modeled as a Work Breakdown Element (i.e., it appears as part of a work breakdown structure and can have predecessors and successors).

2.6.12 Role Use

Super Class

- Breakdown Element (from SPEM2.0 Process Structure)

Description

Role Use is a special Breakdown Element that either represents a performer of an Activity or a participant of the Activity. If it is a performer, the Role Use and Activity need to be related via a Process Performer. If it is a participant, then the Role Use is simply stored in the nestedBreakdownElement composition of the Activity and might be used by one of the sub-activities as a performer and/or a Process Responsibility Assignment. Role Uses are only valid within the context of an Activity. They are not to be reused across activities.

Semantics

A Role Use represents an activity-specific occurrence of an activity performer or participant. A Role Use is an activity-specific object and not a general reusable definition of an organizational role. A Role Use represents the occurrence of a real person performing activity-specific work and having activity-specific responsibilities. A Role Use participant stored with an Activity can be only accessed and reused by the Activity's sub-Activities and not by any parent or sibling Activities in the Activity breakdown structure. This scoping of Role Use in the local namespace of Activities allows different Performers as well as different Responsibility Assignments for every Activity. In other words, Role Use instances with the same name, but different responsibilities and performing different work, can be created in different Activities.

2.6.13 Process Performer

Super Class

- Breakdown Element (from SPEM2.0 Process Structure)
- Work Definition Performer (from CM_SPEM Core)

Description

Process Performer is a Breakdown Element and Work Definition Performer that represents a relationship between Activity instances and Role Use instances. An instance of Process Performer links one or more Role Use instances to one Activity.

Association Properties

- linkedActivity: Activity. A Process Performer links to zero or one Activity. The linked Activity property subsets the linked Work Definition property from the Work Definition Performer defined in Core.
- linkedRoleUse: RoleUse. A Process Performer links to one or more Role Use.

Semantics

The Process Performer links Role Uses to Activities, indicating that these Role Use instances participate in the work defined by the activity in one or another way. The kind of involvement of the Role Use in the Activity needs to be defined by Kind (Section 8.2) class instances that qualify the Process Performer instances. Typical examples for Kinds of Process Performers would be Primary Performer, Additional Performer, Assisting Performer, Supervising Performer, Consulted Performer, etc. The popular RACI-VS diagram defines another set of commonly used Kinds for the Process Performer: Responsible, Accountable, Consulted, Informed, Verifies, and Signs.

2.6.14 Process Responsibility Assignment

Super Class

- Extensible Element (from SPEM2.0 Core)

Description

Process Responsibility Assignment is a Breakdown Element that represents a relationship between instances of Role Use and Work Product Use. An instance of the Process Responsibility Assignment links one or more Role Use instances to exactly one Work Product Use.

Kind class (from Core) instances shall be used to specify the nature of this responsibility assignment.

Association Properties

- linkedRoleUse: RoleUse. A Process Responsibility Assignment links to one or more Role Use.
- linkedWorkProductUse: WorkProductUse. A Process Responsibility Assignment links to exactly one Work Product Use.

Semantics

The Process Responsibility Assignment links Role Uses to Work Product Uses indicating that the Role Use has a responsibility relationship with the Work Product Use. The Process Performer and Process Responsibility represent two quite different sets of information as a Role Use can be involved in an Activity that modifies a work product without being responsible for the Work Product itself and vice versa (i.e. a Role Use can be responsible for a Work Product Use without participating in all the Activities that modify it).

2.7 EXAMPLE OF MDE PROCESS STRUCTURE DESCRIPTION WITH CM_SPEM

This section aims to illustrate the use of CM_SPEM Process concepts through an MDE process example: the UWE (UML-based Web Engineering) process (described in details in [Koch 2006], and [Kroiß 2008]). First, we give a summarized overview of UWE. Then, we show how CM_SPEM concepts may be used for describing the structure of UWE.

2.7.1 UWE Process Overview

The objective of the UWE process is to give to web developers a systematic and semi-automatic support of web systems development based on models and their transformations.

The process covers the whole development life cycle of web systems from the requirements specification to code generation. It is a model-driven development process following the MDA principles and using the OMG's standards. It consists of a set models and model transformations, specified by metamodels and model transformation languages.

The process starts with the definition of a *requirements model* that is computational independent (CIM) business model. Two sets of platform independent design models (PIM) are derived from these requirements: *functional models* which represent the different concerns of the Web system (content, navigation, business logic, presentation, and adaptation); and an *architecture model* which represents the architectural features of the Web system. *Functional models* are afterwards integrated mainly for the purpose of verification into a *big picture model*. A merge of this *big picture model* with the *architectural models* results in an *integrated model* covering functional and architectural aspects. Finally, platform specific models (PSM) are derived from the *integrated model* from which programming code can be generated.

2.7.2 Describing the UWE process with the CM_SPEM Process Structure concepts

Figure 6 shows the set of graphical representations used for describing the structure of the UWE process.

CM_SPEM Process Structure concepts	Icon	Comment
Activity (from CM_SPEM::Activity Structure)		Reused from SPEM2.0
RoleUse (from CM_SPEM::Activity Structure)		Reused from SPEM2.0
WorkProductUse (from CM_SPEM::Activity Structure)		Reused from SPEM2.0
Transformation (from CM_SPEM::Transformation Structure)	T	New CM_SPEM Icon
TypedModel (from CM_SPEM::Transformation Structure)		New CM_SPEM Icon
Metamodel (from CM_SPEM::Model Structure)	MM	New CM_SPEM Icon

Figure 6 - Graphical representation of CM_SPEM Process Structure concepts

Erreur ! Source du renvoi introuvable. shows a description of the UWE process, as an Activity that represents the whole process. The UWE process takes as input the Work Product Use "Web System Requirements" (which may be a text file, for example), and produces as outputs the Work Product Uses "JEE Source Code", and ".NET Source Code". The performers Involved in the process play roles defined by the Role Uses "Web Designer", "JEE Developer", and ".NET Developer".

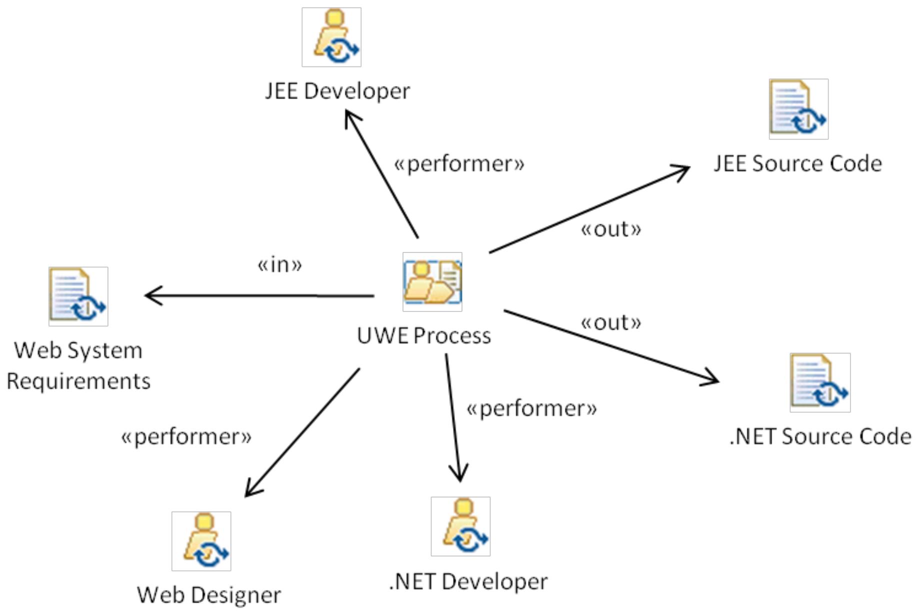

Figure 7 - The whole UWE Process as a single Activity

shows the Breakdown Elements nested by the UWE Process, which are: the Work Product Uses “Web System Requirements”, “JEE Source Code”, “.NET Source Code”; the Role Uses “Web Designer”, “JEE Developer”, and “.NET Developer”; the Activities “Create Requirements Model”, “Generate JEE Source Code”, and “Generate .NET Source Code”; the Transformations “Requirements To Functional”, “Requirements To Architecture”, “Functional To Big Picture”, “Architecture Integration”, “Integration To JEE”, and “Integration To .NET”; the Models “Requirements Model”, “Functional Models”, “Architecture Model”, “Big Picture Model”, “Integration Model”, “Model For JEE”, and “Model For .NET”.

Figure 9 shows the parameters of Activities and Transformations.

Figure 10 shows their precedence relationships (depicted by the «finish to start» associations), and their associated performers.

As shown by Figure 11, the “Requirements To Functional” transformation has six sub-transformations: “Requirements To Content”, “Content To Navigation”, “Requirements To Navigation”, “Navigation Refinement”, “Navigation To Presentation”, and “Style Adjustment”.

Figure 7 shows the precedence links and model parameters of these sub-transformations.

Finally, figure 8 shows the relationships between the models and metamodels involved in the process. The "Functional Models" is a set composed of four models: the "Content Model", the "Navigation Model", the "Presentation Model", and the "Style Guide Model". Conformity relationships are depicted as associations stereotyped with «conform to».

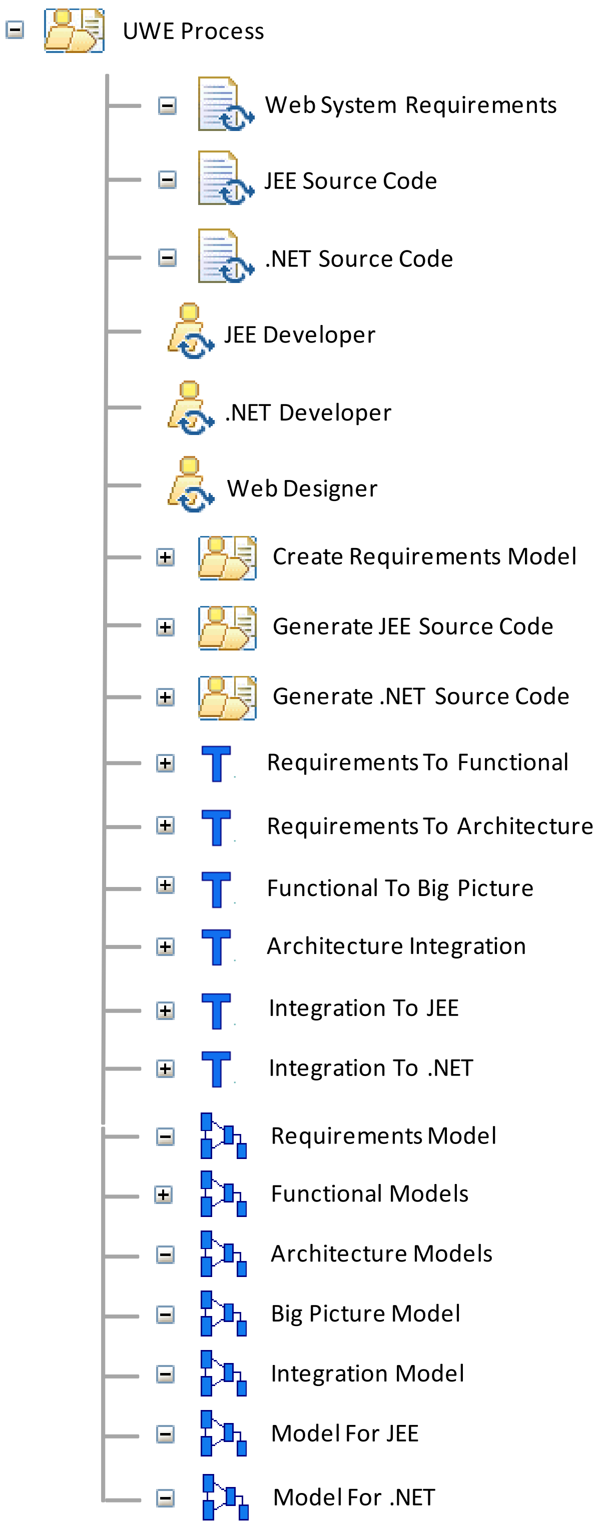

Figure 8 - UWE's Breakdown Elements

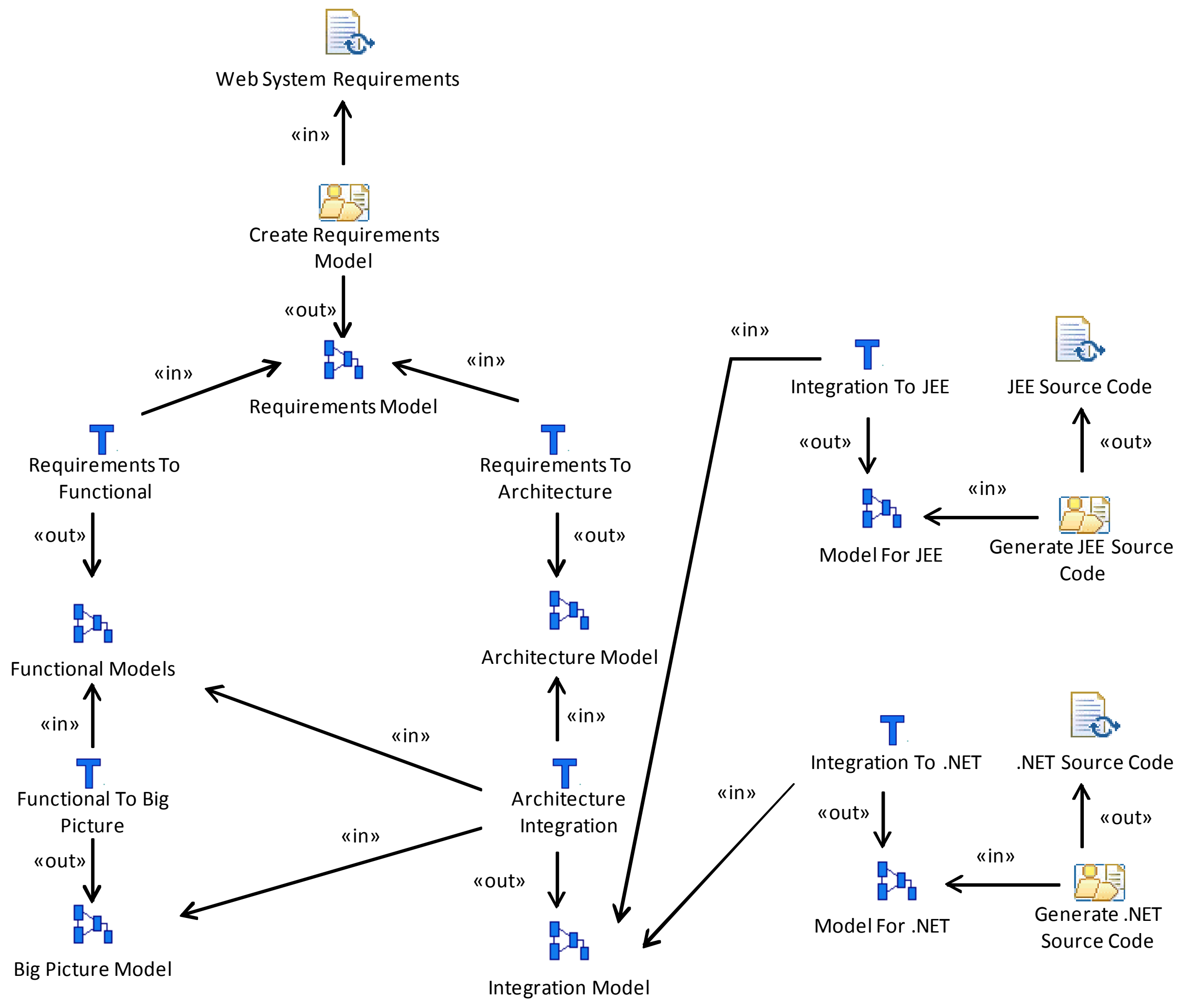

Figure 9 - Parameters of UWE's Activities and Transformations

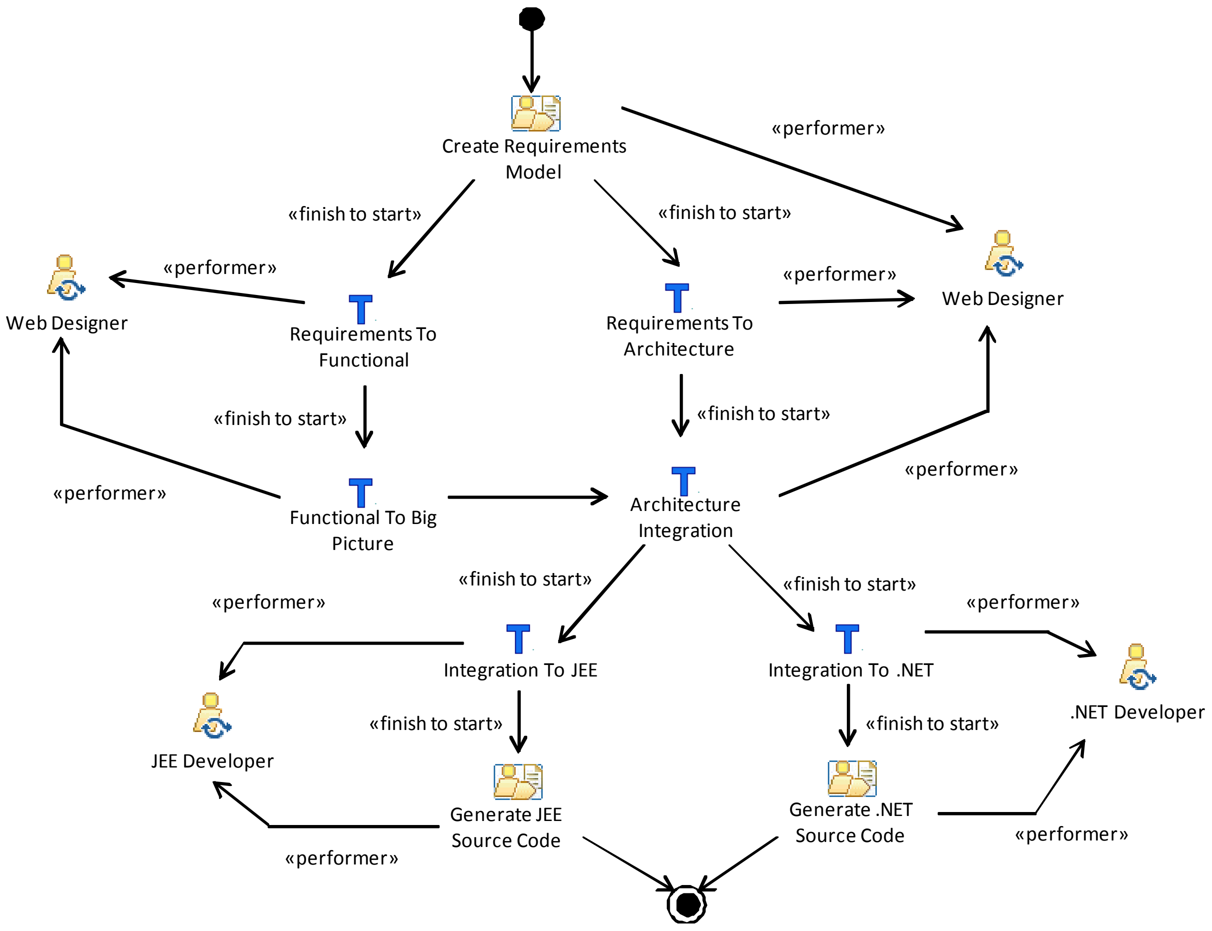

Figure 10 - Precedence links and Performers of UWE's Activities and Transformations

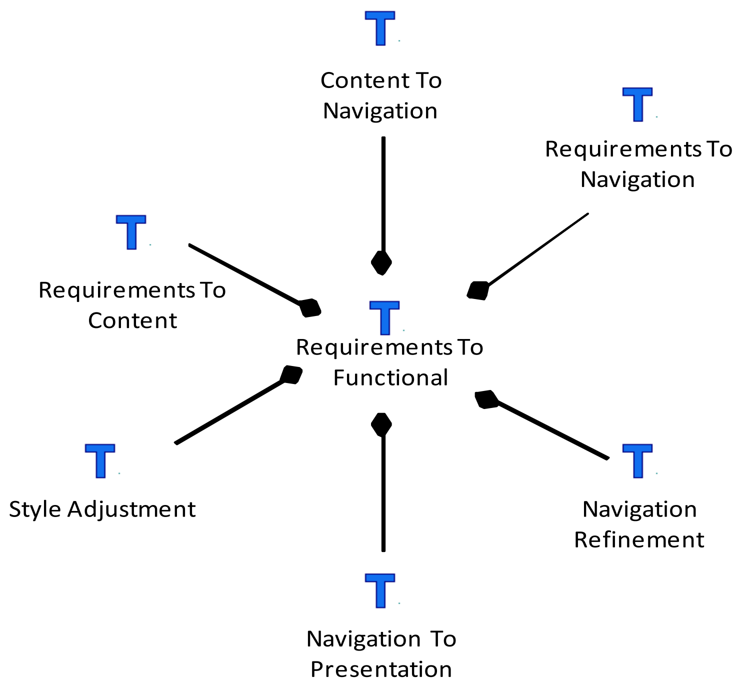

Figure 11 - Sub-transformations of the “Requirements model to functional models” transformation

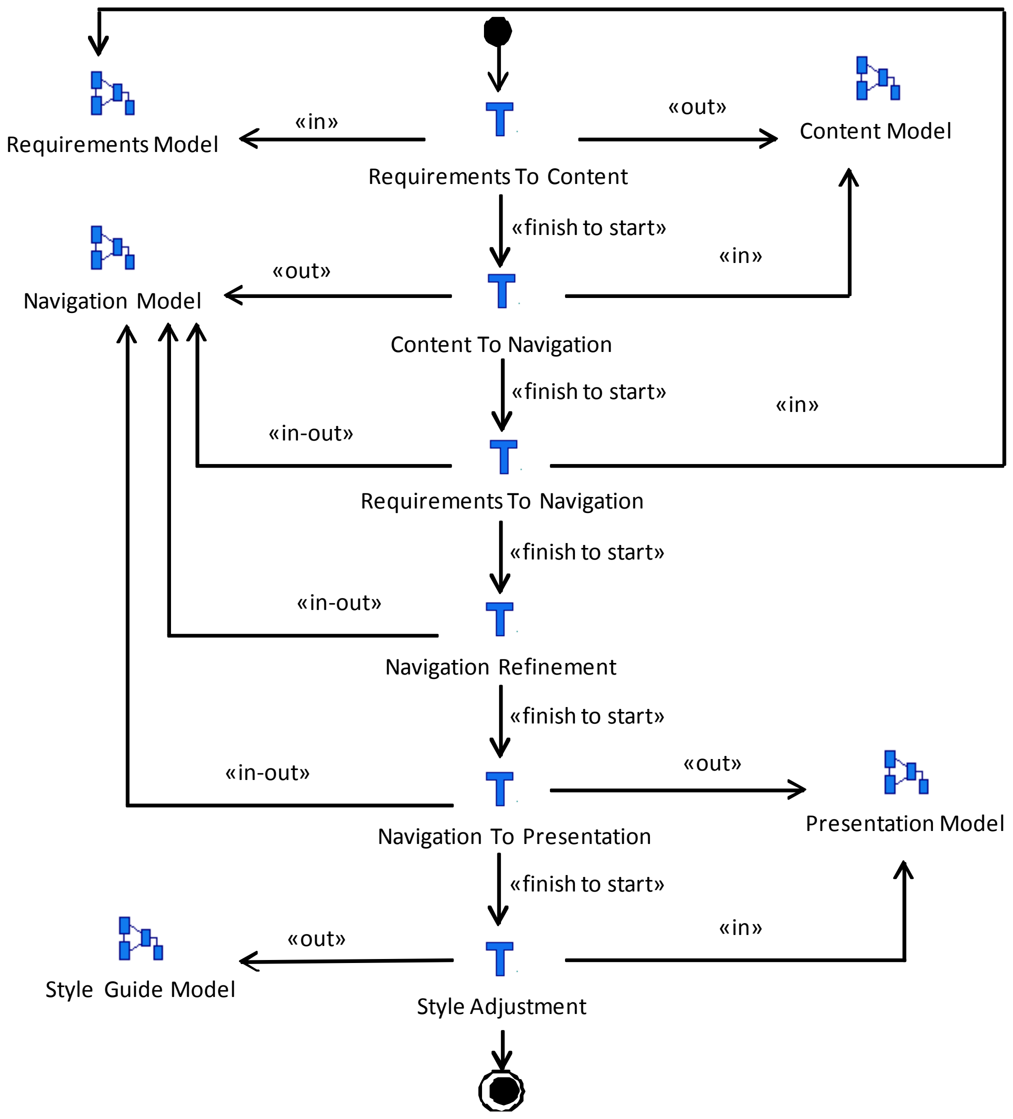

Figure 12 - Precedence links and Parameters of the sub-transformations of the "Requirements model to functional models" transformation

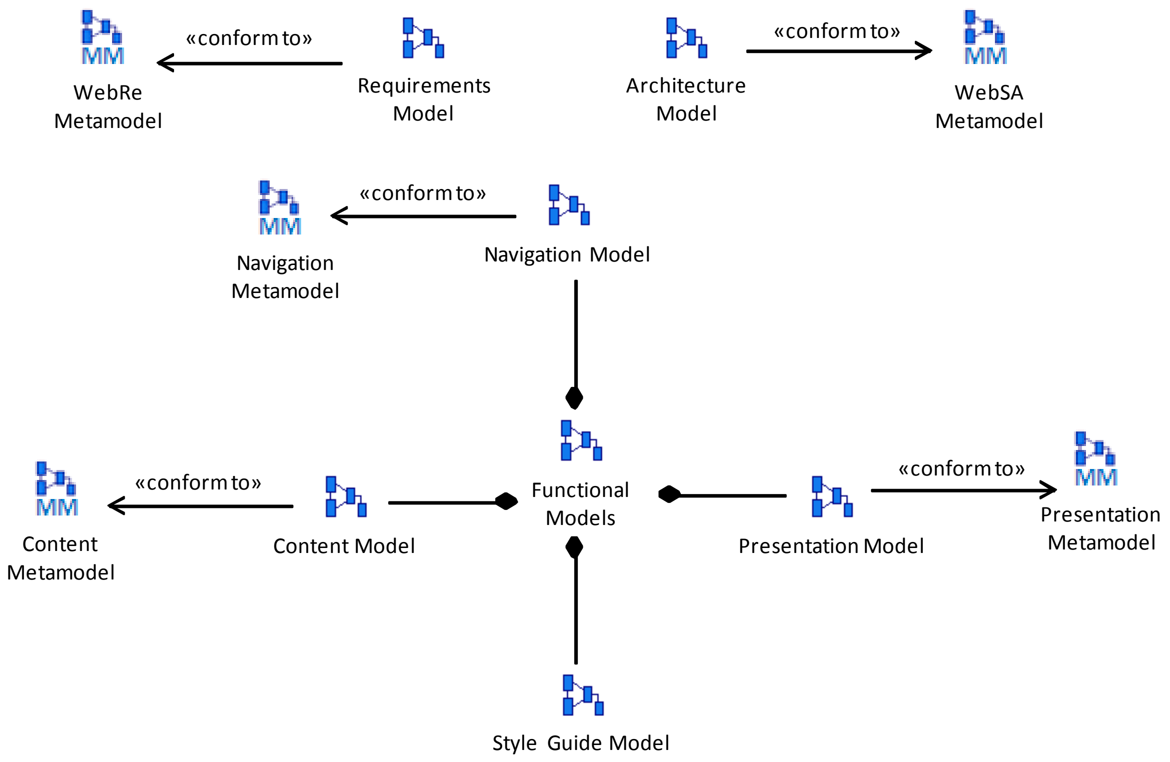

Figure 13 - UWE's models and metamodels relationships

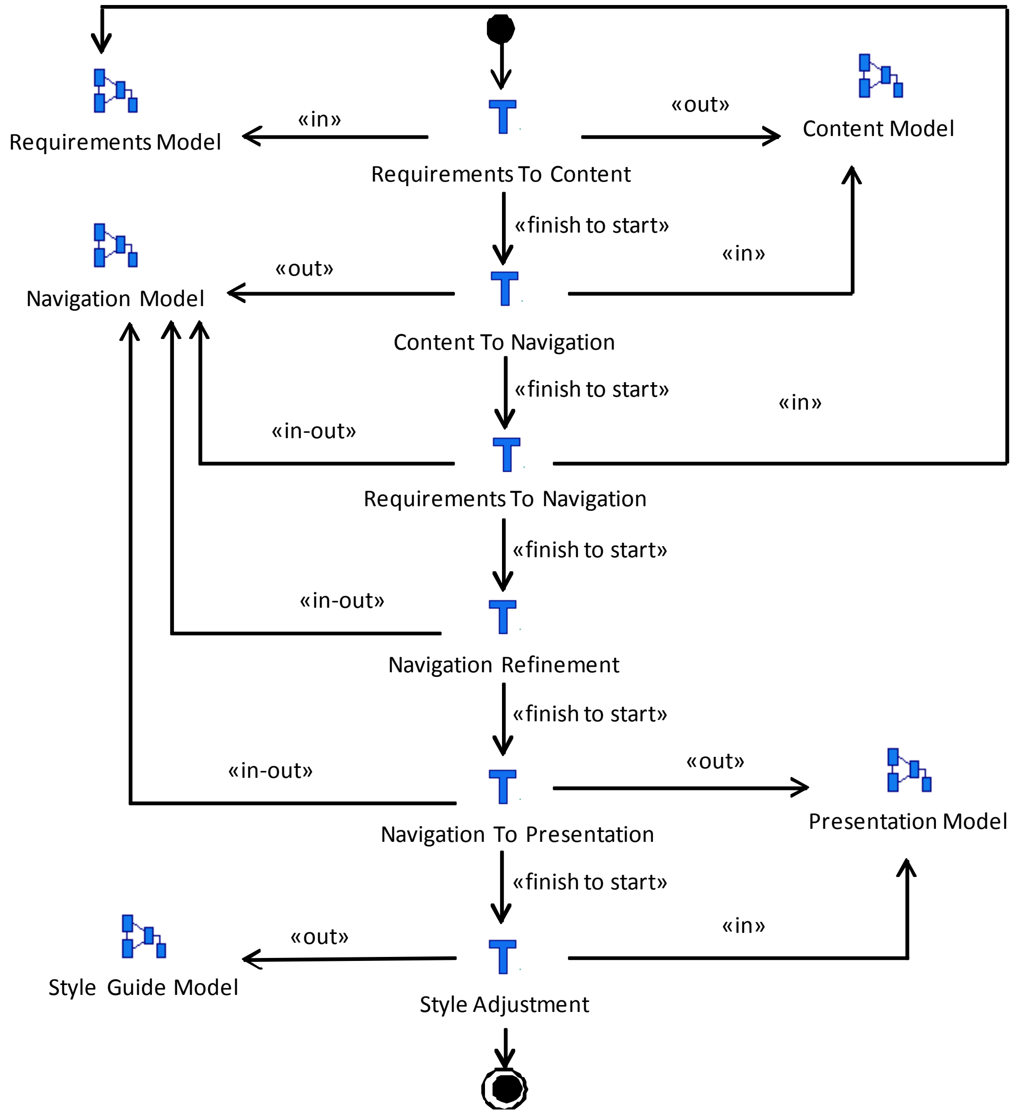

Figure 14 – Precedence links and Parameters of the sub-transformations of the "Requirements model to functional models" transformation

2.8 CONCLUSION

The CM_SPEM Process Structure metamodel reuses a subset of SPEM 2.0 and QVT in order to include concepts related to MDE processes. A model is defined as a specialization of a SPEM work product, thus activities working on models may be described. A model transformation is defined as a specialization of SPEM work definition that has models as input/output parameters, and also as a SPEM work breakdown element. Thus, model transformations may be nested by SPEM activities. As in SPEM2.0, an MDE process is considered as a kind of Activity, which may contain model transformations, and activities that work on models or other kind of work products. All these concepts have been illustrated through the example of the UWE MDE process.

3. COLLABORATIVE MDE PROCESS MODELLING

3.1 INTRODUCTION

The goal of the current section is the definition of a formalism suited for describing collaboration. The approach taken is to reuse the Software & Systems Process Engineering Metamodel (SPEM), and extend its collaboration description facilities.

The extensions proposed are rooted in a fundamental realization: a considerable amount of relations relevant to collaboration can be described only at the project level.

Indeed, most conventional process metamodels (including SPEM) allow defining a process via the activities carried out inside the process. Activities are performed by roles and manipulate products. The defined process then can be enacted in various ways for different projects.

In the context of a specific project, where collaboration occurs, there may be:

- Different actors (people) carrying out the same task. Usually, this is because the same task applies to a large artifact, which can be decomposed into parts assigned to different people. For example, when writing unit tests, each project participant can work on a set of system components. Then, creating test cases for that particular set of component can be considered as an actor specific task (in contrast to the process task of creating all unit tests).
- Different physical artifacts in different workspaces which may stand for the same product. This happens for example when each participant has his own working copy of a shared product (each copy is thus an actor specific artifact). In those cases, some guidelines are needed to designate the reference copy, and how the other copies contribute to it.

However, the relations involved in the above situations cannot be described in conventional process models like SPEM, as the concepts needed (actors, actor specific tasks, actor specific artifacts) are not represented in these process models, but appear only when enacting processes.

To allow the description of such relations, we chose to include the missing concepts in the metamodel. This does mean that part of the process model will be constructed only after some pieces of information are available (like the people working together on a shared product). But it is still useful to do so, as being explicit about these relations makes some tool assistance possible. For example, a tool can automatically inform the relevant people when some modification is made to a physical artifact.

The integration of afore mentioned concepts make possible the description of the following relations:

- Between actors manipulating the same product or playing the same role
- Between actor specific tasks carried out in the same (process) task or on the same product
- Between actor specific artifacts that represent the same product or are manipulated in the same (process) task
- Between actor specific artifacts and actor specific tasks, actor specific tasks and actors, actor specific artifacts and actors

Section 3.2 presents an illustrating example of collaboration. Section 3.3 presents the Collaboration Structure package, and section 3.4 models the example using the concepts defined in section 3.3.

3.2 EXAMPLE

We illustrate the additional collaboration description capabilities granted by the new concepts on a set of situations in the same project. This section gives a general overview of the project and the situations, and section 3.4 models the various situations with CM_SPEM.

We consider a development project, with the following participants: Alice, Arthur, Bob, Mike, and Tracy. These actors are the only elements implicitly reused across the situations (those situations have no other implicit link between them otherwise).

3.2.1 Situation 1: The same task carried out on copies of the same artifact

For the task “Elaborate Use Case Model”, four different use cases (UC1, UC2, UC3, and UC4) have been identified by previous tasks. The output of the task is the product “Use Case Model”, which should contain all the four use cases. Alice is asked to elaborate UC1, Bob works on UC2, and Mike works on UC3 and UC4. Alice has the responsibility of assembling those contributions. The input of the task is a requirement document, which is not shown for simplicity, as it is not modified in the task, and is only used as documentation.

In their respective workspaces, each of Alice, Bob, and Mike works on a full copy of the use case model. In other words, at a certain frequency left at the discretion of the designers, Alice (who acts as an integrator) grabs everyone’s contribution, integrates it with the model in her workspace, and then send the whole (updated) model to the other developers. Each developer then continues his work (containing, as much as possible, his modifications to the use case assigned to his/her) from the latest version made available by Alice. At the end of the task, each workspace should have the same use case model, which corresponds to the output of the task.

3.2.2 Situation 2: The same task carried out on distinct artifacts (variation of situation 1)

We start from the same setup as situation 1. But this time, the artifacts manipulated by Bob (UC2), and Mike (UC3 and UC4) are only part of the final “Use Case Model”. In other words, at no moment does Bob have in his workspace UC1, UC3, nor UC4 (idem. for Mike). Alice still acts as integrator, and the artifact in her workspace is the whole model.

This setup reduced the chances that someone (other than Alice the integrator) accidentally modifies a use case he is not responsible for. The case arises when neither Bob, nor Mike needs the other use cases to give context to his own work or needs to worry about the consistency of his use case with the other use cases (when extending or including another use case for example).

Note that in this situation, the artifacts manipulated by Bob and Mike can be viewed as temporary artifacts, and need not be preserved at the end of the activity. They have already been integrated in the artifact manipulated by Alice, which represents the output of the task, the whole “Use Case Model”. In a real world scenario, this can be accomplished using the branching feature of a version control system.

3.2.3 Situation 3: A task carried out by people playing different roles

Bob is writing unit tests (the corresponding product is "Unit tests") for a component that implements functionality needed by the above use cases UC1 and UC2. Tracy will do a code review on the tests written by Bob. The code review is done periodically, before each product release. When reviewing the code, Tracy modifies the artifacts in her workspace, and when done, notifies Bob so he can pull the changes into his workspace, and base further work on that version.

3.3 THE COLLABORATION STRUCTURE PACKAGE

3.3.1 General overview

The collaboration aspects of CM_SPEM are described in the CM_SPEM::CollaborationStructure package. The main concepts this package borrows from SPEM are RoleUse (from SPEM::ProcessStructure via CM_SPEM::ProcessStructure), ProductUse (from SPEM::ProcessStructure via CM_SPEM::ProcessStructure) and TaskUse (from SPEM::ProcessWithMethods). The main concepts introduced are Actor, ActorSpecificTask, and ActorSpecificArtifact. The other concepts are mostly relations between the new concepts, or between concepts form SPEM and the newly defined concepts.

Figure 15 shows which SPEM packages are reused by CM_SPEM::CollaborationStructure, and Figure 16 is a high level overview of the content of the CM_SPEM::CollaborationStructure package.

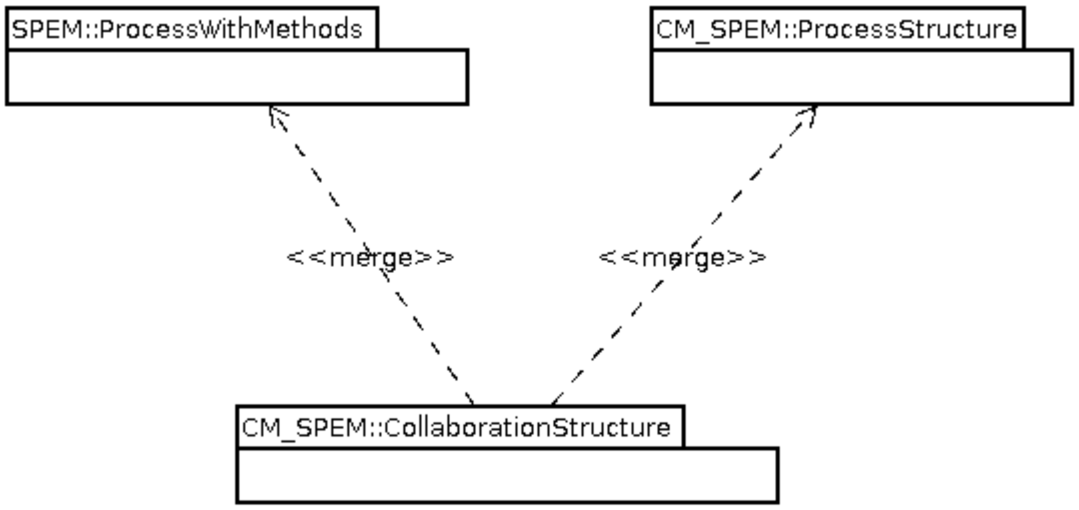

Figure 15 - SPEM Packages merged by the CM_SPEM::CollaborationStructure package

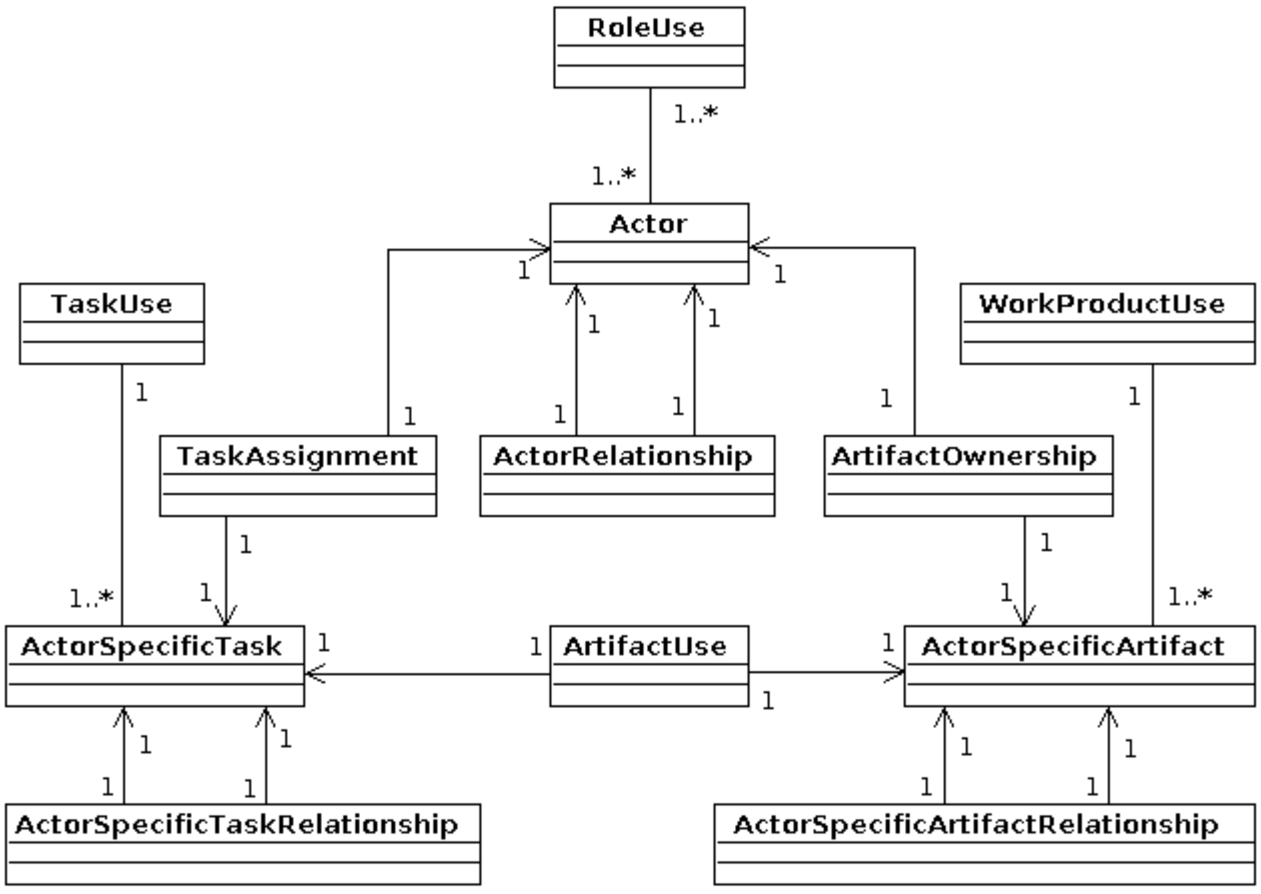

Figure 16 - Overview of the CM_SPEM::CollaborationStructure package

3.3.2 Newly introduced concepts

3.3.2.1 Actor

Super Class

ExtensibleElement (SPEM::Core)

Description

Actor is an ExtensibleElement that represents a specific human participant in a project.

Attributes

Association properties

- associatedRoleUse: RoleUse. This association represents a resource affectation, that is, the affectation of an actor to a defined RoleUse. The set of all associatedRoleUse for a given actor gives an overview of where he/she contributes in a project.

Semantics

An Actor is a specific human affected to a RoleUse when a project is enacted. An Actor unambiguously identifies a single person in a project, and as such, is scoped to the whole process model.

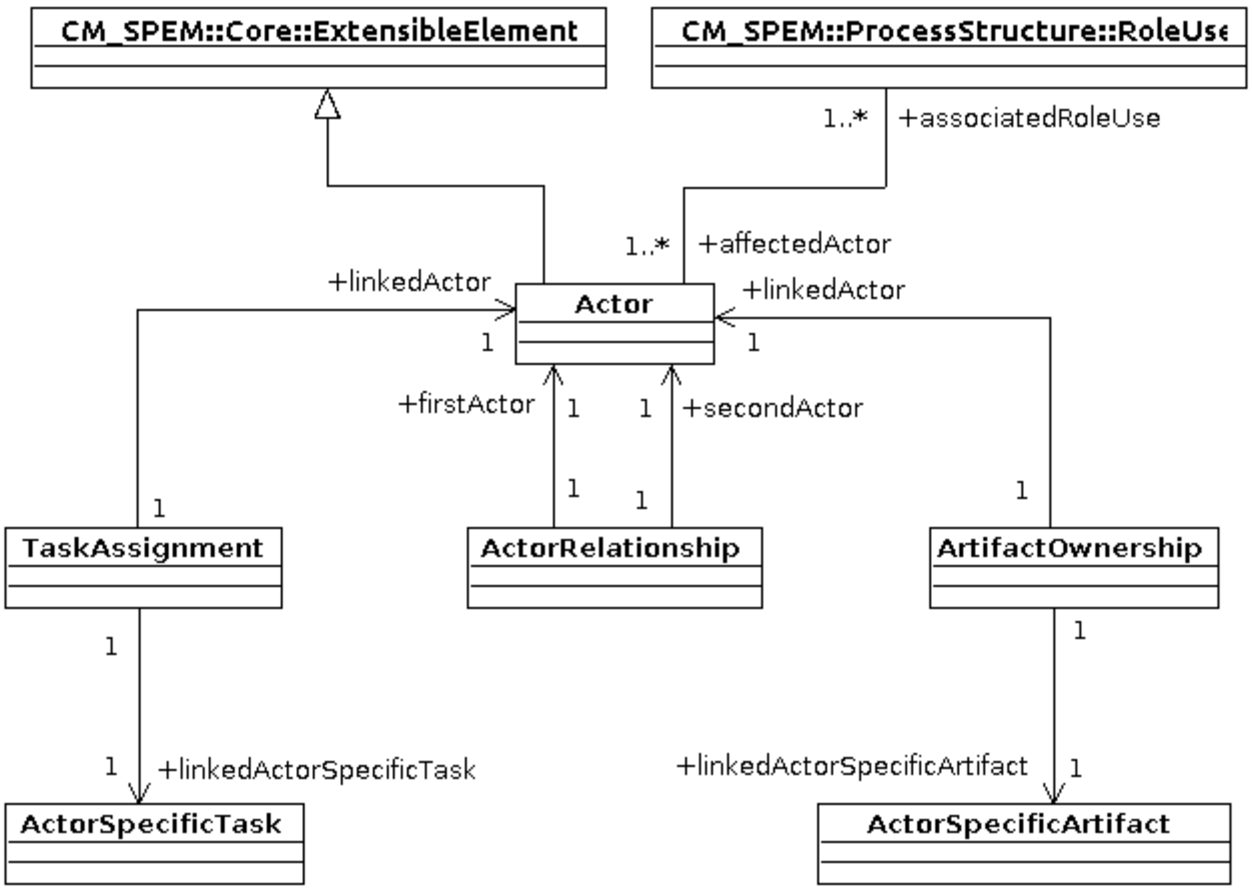

Figure 17 - Actor and linked concepts in the CM_SPEM::CollaborationStructure package

3.3.2.2 ActorSpecificTask

Super Class

ExtensibleElement (SPEM::Core)

WorkBreakDownElement (SPEM::ProcessStructure)

Description

ActorSpecificTask represents the work done by a single actor in the context of a specific TaskUse.

Attributes

Association properties

- associatedTaskUse: TaskUse. This association represents the contribution of an ActorSpecificTask to a TaskUse. It means the ActorSpecificTask is part of the work required for the associated TaskUse.

Semantics

An ActorSpecificTask is a unit of work done by a specific actor, towards the execution of a TaskUse. For example, when the steps required by a TaskUse need to be repeated for three different components, and when each component is assigned to a different Actor, three different ActorSpecificTask will be created, each for one Actor manipulating a component.

An ActorSpecificTask is scoped to the TaskUse it contributes to.

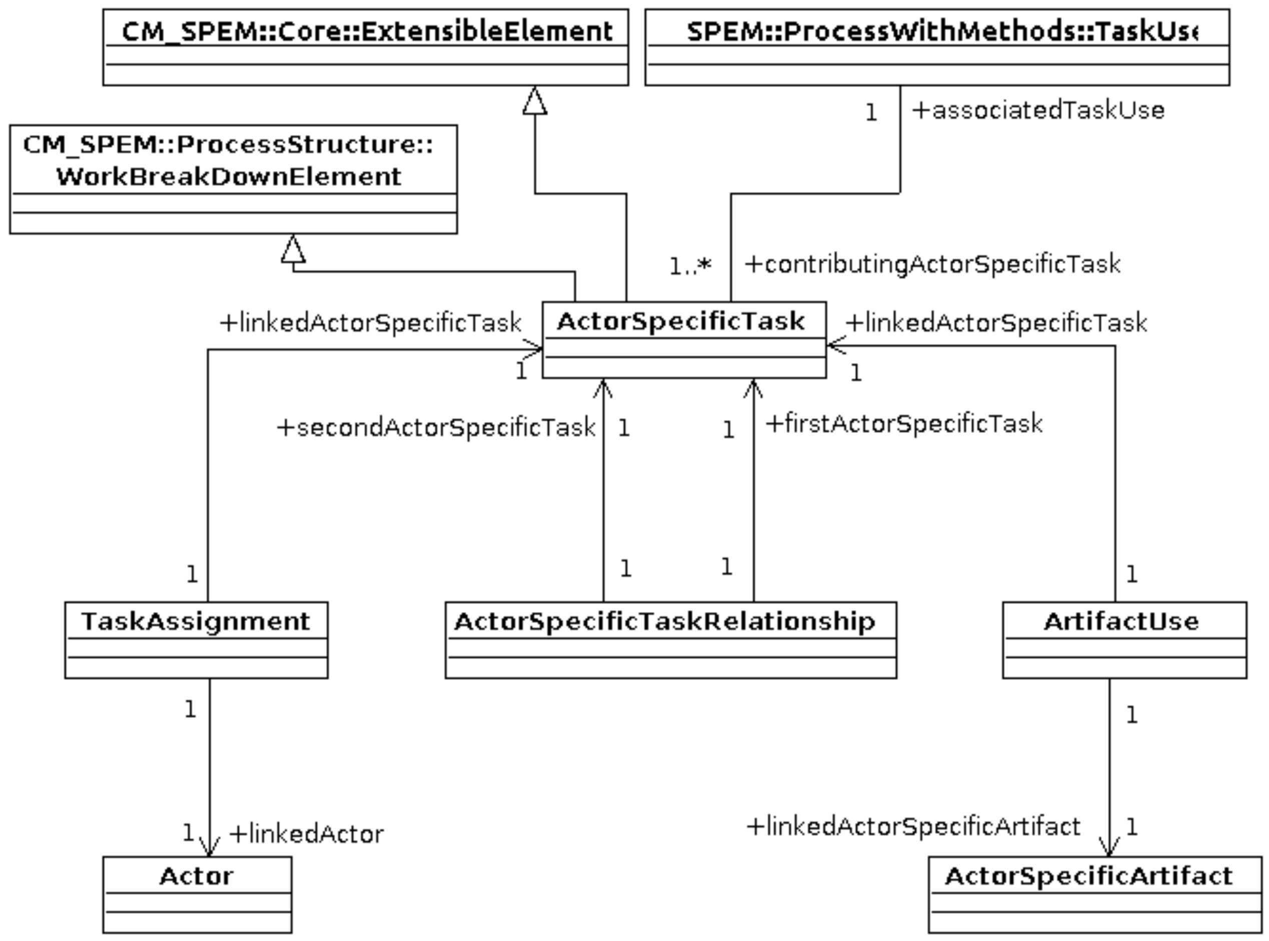
CM_SPEM::Core::ExtensibleElement
SPEM::ProcessWithMethods::TaskUse
CM_SPEM::ProcessStructure::WorkBreakDownElement
1 +associatedTaskUse
1..* +contributingActorSpecificTask
+linkedActorSpecificTask 1
ActorSpecificTask
+linkedActorSpecificTask 1
+secondActorSpecificTask 1
1 +firstActorSpecificTask
1
1
1
1
TaskAssignment
ActorSpecificTaskRelationship
ArtifactUse
1
1
1 +linkedActor
Actor
+linkedActorSpecificArtifact 1
ActorSpecificArtifact

Figure 18 - ActorSpecificTask and linked concepts in the CM_SPEM::CollaborationStructure package

3.3.2.3 ActorSpecificArtifact

Super Class

ExtensibleElement (SPEM::Core)

Description

ActorSpecificArtifact represents a copy of a WorkProductUse in an actor's workspace.

Attributes

- isPartialCopy: Boolean = false. When set to true, this attribute indicates that the ActorSpecificArtifact is only a partial copy of the corresponding WorkProductUse. This is handy when a WorkProductUse has sufficiently autonomous part, so as to allow different persons to work relatively independently on each part. Each part is then an ActorSpecificArtifact with isPartialCopy is set to true. There is an implicit composition step (and an eventual prior decomposition step) where the various parts are brought together, to form an ActorSpecificArtifact which is a full representation of the WorkProductUse.

Association properties

- associatedWorkProductUse: WorkProductUse. This association represents the representation of a WorkProductUse by an ActorSpecificArtifact. It means that the ActorSpecificArtifact is one of the copies of the associated WorkProductUse.

Semantics

An ActorSpecificArtifact is an occurrence of a WorkProductUse, in the personal workspace of a specific actor. This is the personal copy of the actor, and is manipulated only by him/her. An ActorSpecificArtifact is scoped to the WorkProductUse it represents.

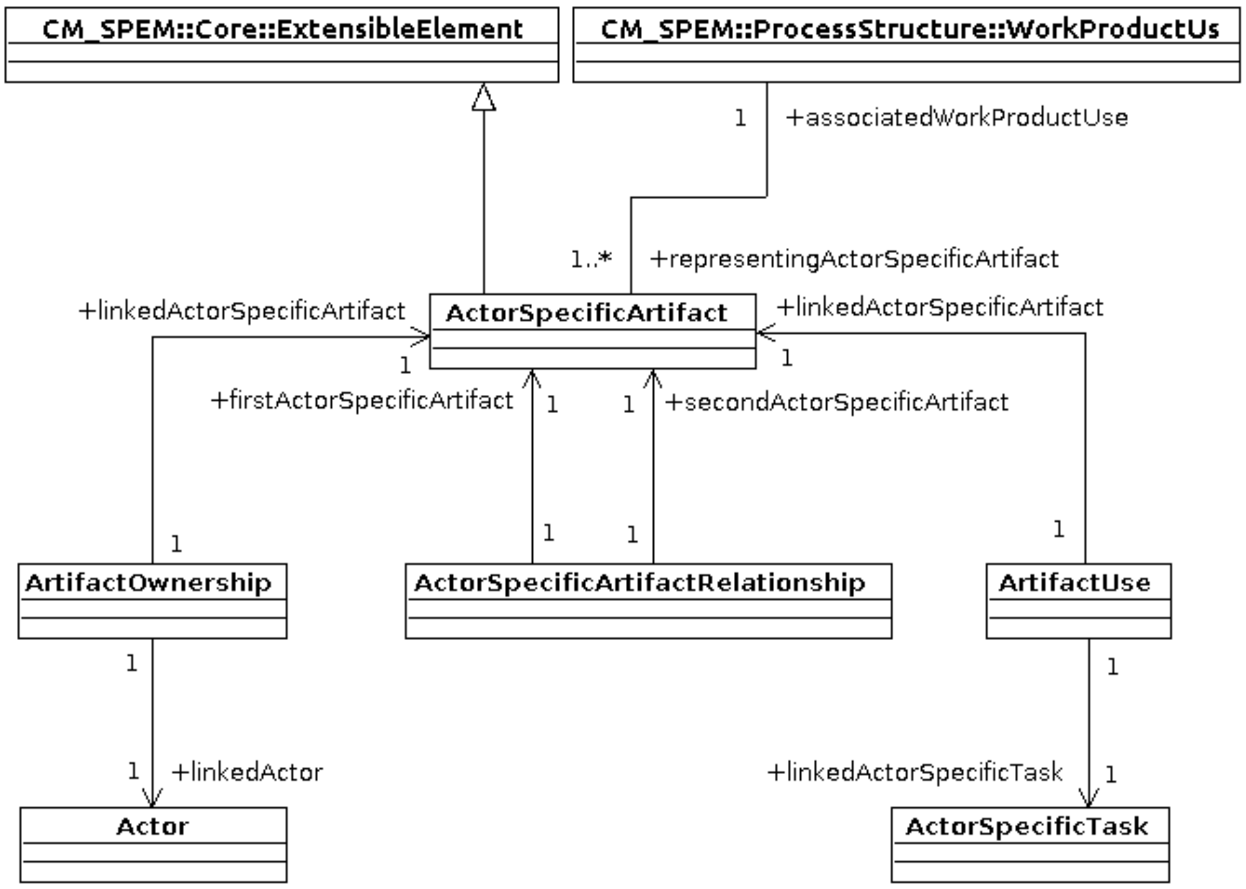
CM_SPEM::Core::ExtensibleElement
CM_SPEM::ProcessStructure::WorkProductUs
1
+associatedWorkProductUse
1..*
+representingActorSpecificArtifact
+linkedActorSpecificArtifact
ActorSpecificArtifact
+linkedActorSpecificArtifact
1
1
+firstActorSpecificArtifact
1
1
+secondActorSpecificArtifact
1
1
1
1
ArtifactOwnership
ActorSpecificArtifactRelationship
ArtifactUse
1
1
1
+linkedActor
+linkedActorSpecificTask
1
Actor
ActorSpecificTask

Figure 19 - ActorSpecificArtifact and linked concepts in the CM_SPEM::CollaborationStructure package

3.3.2.4 ActorRelationship

Super Class

BreakdownElement (SPEM::ProcessStructure)

ExtensibleElement (SPEM::Core)

Description

Actor Relationship represents a special link between two actors (Figure 17).

Attributes

Association properties

- firstActor: Actor. The first Actor involved in the relation.
- secondActor: Actor. The second Actor involved in the relation.

Semantics

The kind of relationships between the two actors is specified by associating a subclass of ActorRelationshipKind (see section 3.3.4.1) to the instance of ActorRelationship.

3.3.2.5 ActorSpecificTaskRelationship

Super Class

BreakdownElement (SPEM::ProcessStructure)

ExtensibleElement (SPEM::Core)

Description

ActorSpecificTaskRelationship represents a special link between two tasks, in the context of the TaskUse they contribute to (Figure 18).

Attributes

Association properties

- firstActorSpecificTask: ActorSpecificTask. The first ActorSpecificTask involved in the relation.
- secondActorSpecificTask: ActorSpecificTask. The second ActorSpecificTask involved in the relation.

Semantics

The kind of relationships between the two actor specific tasks is specified by associating a subclass of ActorSpecificTaskRelationshipKind (see section 3.3.4.2) to the instance of ActorSpecificTaskRelationship.

3.3.2.6 ActorSpecificArtifactRelationship

Super Class

BreakdownElement (SPEM::ProcessStructure)

ExtensibleElement (SPEM::Core)

Description

ActorSpecificArtifactRelationship is a special link between two ActorSpecificArtifacts, in the context of the WorkProductUse they represent (Figure 19).

Attributes

Association properties

- firstActorSpecificArtifact: ActorSpecificArtifact. The first ActorSpecificArtifact involved in the relation.
- secondActorSpecificArtifact: ActorSpecificArtifact. The second ActorSpecificArtifact involved in the relation.

Semantics

The kind of relationships between the two actor specific artifacts is specified by associating a subclass of ActorSpecificArtifactRelationshipKind (see section 3.3.4.3) to the instance of ActorSpecificArtifactRelationship.

It should be noted that this relationship should only be used for relation specific to two instances of ActorSpecificArtifact. If such relation is applicable to any couple of ActorSpecificArtifact created from the related couple of WorkProductUse, then it is better to use a WorkProductUseRelationship (SPEM::ProcessStructure), a WorkProductDefinitionRelationship (SPEM::MethodContent), or a WorkSequence (SPEM::ProcessStructure) instead.

3.3.2.7 TaskAssignment

Super Class

BreakdownElement (SPEM::ProcessStructure)

ExtensibleElement (SPEM::Core)

Description

TaskAssignment represents a relationship between an ActorSpecificTask and the Actor that carries it out.

Attributes

Association properties

- linkedActor: Actor. The Actor a task is being assigned to.
- linkedActorSpecificTask: ActorSpecificTask. The task being assigned to an actor.

Semantics

A TaskAssignment links an ActorSpecificTask to the Actor assigned to it. This adds a useful precision when a TaskUse instance corresponds to more than one ActorSpecificTask instances. Even when a TaskUse corresponds to only one ActorSpecificTask, the relationship TaskAssignment adds a precision about the Actor assigned to the ActorSpecificTask (i.e. to the TaskUse), which cannot not be specified with a WorkDefinitionPerformer (SPEM::MethodContent) or a ProcessPerformer (SPEM::ProcessStructure).

When a TaskUse has only one ActorSpecificTask, and a RoleUse is affected to only one Actor, and a ProcessPerformer already connects the TaskUse and the RoleUse, then a TaskAssignment relation between the ActorSpecificTask and the Actor is a direct consequence, and may be omitted for brevity’s sake. This holds when the ProcessPerformer is replaced by a Performer.

Constraints

- Whenever a TaskAssignment links an Actor and an ActorSpecificTask, the corresponding TaskUse and TaskUse should be related with a ProcessPerformer (SPEM::ProcessStructure).

3.3.2.8 ArtifactUse

Super Class

BreakdownElement (SPEM::ProcessStructure)

ExtensibleElement (SPEM::Core)

Description

ArtifactUse represents a relationship between an ActorSpecificTask and an ActorSpecificArtifact that is manipulated when carrying it out.

Attributes

Association properties

- linkedActorSpecificTask: ActorSpecificTask. The ActorSpecificTask an artifact is manipulated in.
- linkedActorSpecificArtifact: ActorSpecificArtifact. The ActorSpecificArtifact a task manipulates.

Semantics

An ArtifactUse connects an ActorSpecificArtifact to an ActorSpecificTask, and states that the artifact is manipulated in the context of that particular ActorSpecificTask. It should be noted that this implies that the Actor the ActorSpecificTask is assigned to has an ArtifactOwnership relation with the ActorSpecificArtifact (the reverse is not always true).

ArtifactUse differs from WorkDefinitionParameter (SPEM::Core), ProcessParameter (SPEM::ProcessPerformer), and DefaultTaskDefinitionParameter (SPEM::MethodContent) by virtue of being specific, not to a WorkProductUse, but to one of its physical representations (that is, an ActorSpecificArtifact).

When a TaskUse has only one ActorSpecificTask, and a WorkProductUse is represented by only one ActorSpecificArtifact, and a ProcessParameter already connects the TaskUse and the WorkProductUse, then an ArtifactUse relation between the ActorSpecificTask and the ActorSpecificArtifact is a direct consequence, and may be omitted for brevity's sake. This holds when the ProcessParameter is replaced by a WorkDefinitionParameter or a DefaultTaskDefinitionParameter.

Constraints

- Whenever an ActorSpecificTask and an ActorSpecificArtifact are related with an ArtifactUse, the corresponding TaskUse and WorkProductUse should be related with a ProcessPerformer (SPEM::ProcessStructure).

3.3.2.9 ArtifactOwnership

Super Class

BreakdownElement (SPEM::ProcessStructure)

ExtensibleElement (SPEM::Core)

Description

ArtifactOwnership represents a relationship between an Actor and an ActorSpecificArtifact that belongs to his workspace.

Attributes

Association properties

- linkedActor: Actor. The actor that owns an artifact.
- linkedActorSpecificArtifact: ActorSpecificArtifact. The artifact owned by an Actor.

Semantics

An ArtifactOwnership links an Actor to an ActorSpecificArtifact that is part of his workspace. It should be noted that this does not say anything about the ActorSpecificTask the artifact is manipulated in, as an instance of Actor can be assigned to several instances of ActorSpecificTask.

ArtifactOwnership differs from ProcessResponsibilityAssignment (SPEM::ProcessStructure) and DefaultResponsibilityAssignment (SPEM::MethodContent) by virtue of connecting an Actor (not a RoleUse) and an ActorSpecificArtifact (not a WorkProductUse).

When a RoleUse has only one Actor affected to it, and a WorkProductUse is represented by only one ActorSpecificArtifact, and a ProcessResponsibilityAssignment already connects the RoleUse and the WorkProductUse, then an ArtifactOwnership relation between the Actor and the ActorSpecificArtifact is a direct consequence, and may be omitted for brevity's sake. This holds when the ProcessResponsibilityAssignment is replaced by a DefaultResponsibilityAssignment.

Constraints

- Whenever an ActorSpecificArtifact and an Actor are related with an ArtifactOwnership, the corresponding WorkProductUse and RoleUse should be related with a ProcessResponsibilityAssignment (SPEM::ProcessStructure).

3.3.3 Concepts from SPEM

3.3.3.1 RoleUse

RoleUse from SPEM::ProcessStructure is extended with additional association properties.

Association properties

- affectedActor: Actor. This denotes an actor affected to this RoleUse.

Constraints

- If a RoleUse is associated with more than one Actor, then its hasMultipleOccurences attribute must be set to true.

3.3.3.2 TaskUse

TaskUse from SPEM::MethodContent is extended with additional association properties.

Association properties

- contributingActorSpecificTask: ActorSpecificTask. This denotes an ActorSpecificTask which contributes to the TaskUse.

Constraints

- If a TaskUse is associated with more than one ActorSpecificTask, than its hasMultipleOccurences attribute must be set to true.

3.3.3.3 WorkProductUse

WorkProductUse from SPEM::ProcessStructure is extended with additional association properties.

Association properties

- representingActorSpecificArtifact: ActorSpecificArtifact. This denotes an ActorSpecificArtifact which is copy of the WorkProductUse

Constraints

- If a WorkProductUse is associated to more than one ActorSpecificArtifact, than its hasMultipleOccurences attribute must be set to true.

3.3.4 The CM_SPEM Base plug-in

Drawing inspiration from the SPEM 2.0 Base Plug-in, the CM_SPEM Base plug-in is a pre-defined method plug-in, which provides some common instances for CM_SPEM relationships, in the domain of collaboration in software and systems projects. It is only meant to provide a starting point for process modelers.

The plug-in defines three subclasses of SPEM::Core::Kind, which all default and user-defined relationships are instance of.

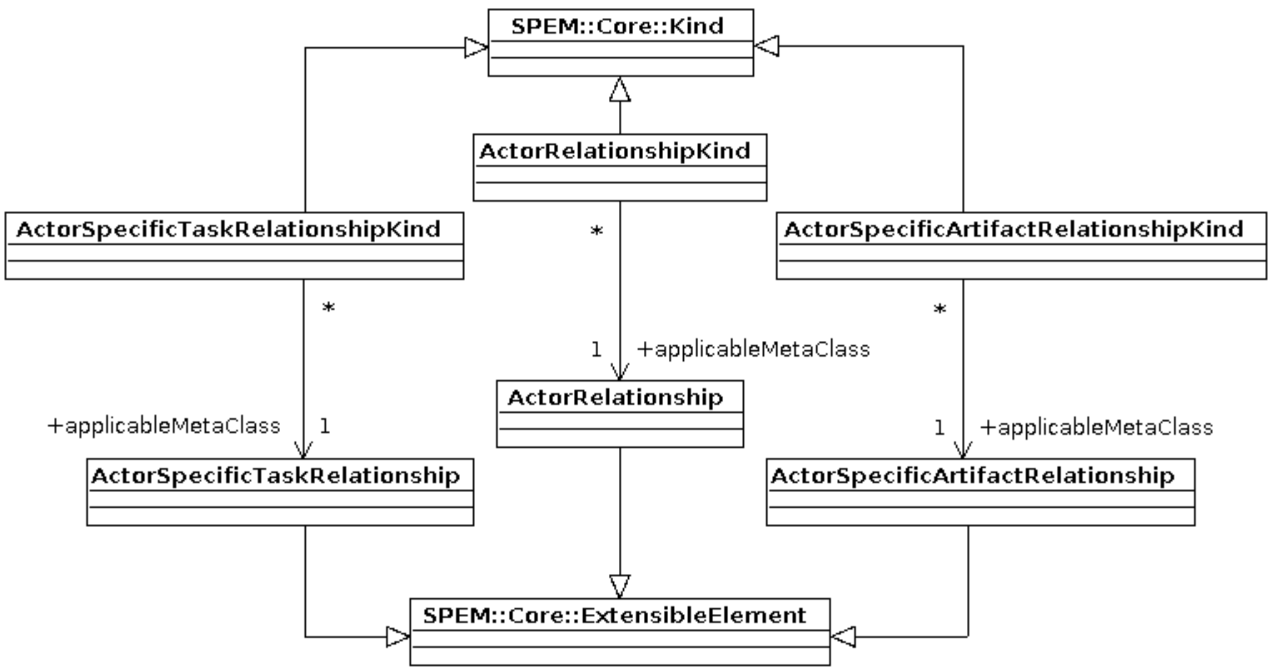
SPEM::Core::Kind
ActorRelationshipKind
ActorSpecificTaskRelationshipKind
ActorSpecificArtifactRelationshipKind
*
1
+applicableMetaClass
ActorRelationship
ActorSpecificTaskRelationship
ActorSpecificArtifactRelationship
SPEM::Core::ExtensibleElement

Figure 20 - The CM_SPEM Base Plugin MetaClasses

3.3.4.1 ActorRelationshipKind

ActorRelationshipKind is a subclass of SPEM ::Core ::Kind, with its applicableMetaClass association end set to ActorRelationship. Its instances are used to qualify relationships between two instances of the Actor metaclass.

The following instances of ActorRelationshipKind are available by default:

- DefaultPushReceiver: Specifies that, by default, the changes the firstActor makes in his workspace are sent to the secondActor.
- DefaultPullSource: Specifies that, by default, the firstActor updates his workspace by pulling changes from the secondActor's workspace.

3.3.4.2 ActorSpecificTaskRelationshipKind

ActorSpecificTaskRelationshipKind is a subclass of SPEM ::Core ::Kind, with its applicableMetaClass association end set to ActorSpecificTaskRelationship. Its instances are used to qualify relationships between two instances of the ActorSpecificTask metaclass.

The following instances of ActorSpecificTaskRelationshipKind are available by default:

- Impact: Specifies that the each time a step is carried out in the firstActorSpecificTask, it potentially requires to be taken into account in the execution of the secondActorSpecificTask.

3.3.4.3 ActorSpecificArtifactRelationshipKind

ActorSpecificArtifactRelationshipKind is a subclass of SPEM::Core::Kind, with its applicableMetaClass association end set to ActorSpecificArtifactRelationship. Its instances are used to qualify relationships between two instances of the ActorSpecificArtifact metaclass.

The following instances of ActorSpecificArtifactRelationshipKind are available by default:

- BlessedCopy: Specifies that, at any moment, the firstActorSpecificArtifact is the authoritative version compared to the secondActorSpecificArtifact.

3.4 MODELISATION OF THE EXAMPLE WITH CM_SPEM

This section shows example models based on CM_SPEM::Collaboration structure, and implementing the situations discussed in section 3.2. The icons defined in SPEM2.0 are reused, and correspondence of the new ones with the new concepts is available below (icons defined in SPEM are also recalled at the end). Relationships are represented by segments when they link instances of different entities (Like an ActorSpecificArtifact and an Actor), and single-headed arrows when they link instances of the same entity (the arrow points to the 'second' instance, as in 'secondActor'). The name of the specific relationship is written inside square brackets on the arrow, like this "[Blessed Copy]".

Actor

ActorRelationship

ActorSpecificTask

ActorSpecificTaskRelationship

ActorSpecificArtifact

ActorSpecificArtifact with isPartialCopy set to true

ActorSpecificArtifactRelationship

ArtifactUse

ArtifactOwnership

TaskAssignment

RoleUse (from SPEM)

TaskUse (from SPEM)

WorkProductUse (from SPEM)

3.4.1 Situation 1: The same task carried out on copies of the same artifact

Three different instances of ActorSpecificTask are created:

- AliceAST in which UC1 is elaborated, using the ActorSpecificArtifact instance Alice_ASA.
- BobAST in which UC2 is elaborated, using the ActorSpecificArtifact instance Bob_ASA
- MikeAST in which UC3 and UC4 are elaborated, using the ActorSpecificArtifact instance Mike_ASA

To keep the diagram clear, links between RoleUse and TaskUse, TaskUse and WorkProductUse, RoleUse and WorkProductUse, have not been drawn. The kinds applied to instances of ActorSpecificArtifactRelationship are indicated by bracketed names (like "[BlessedCopy]").

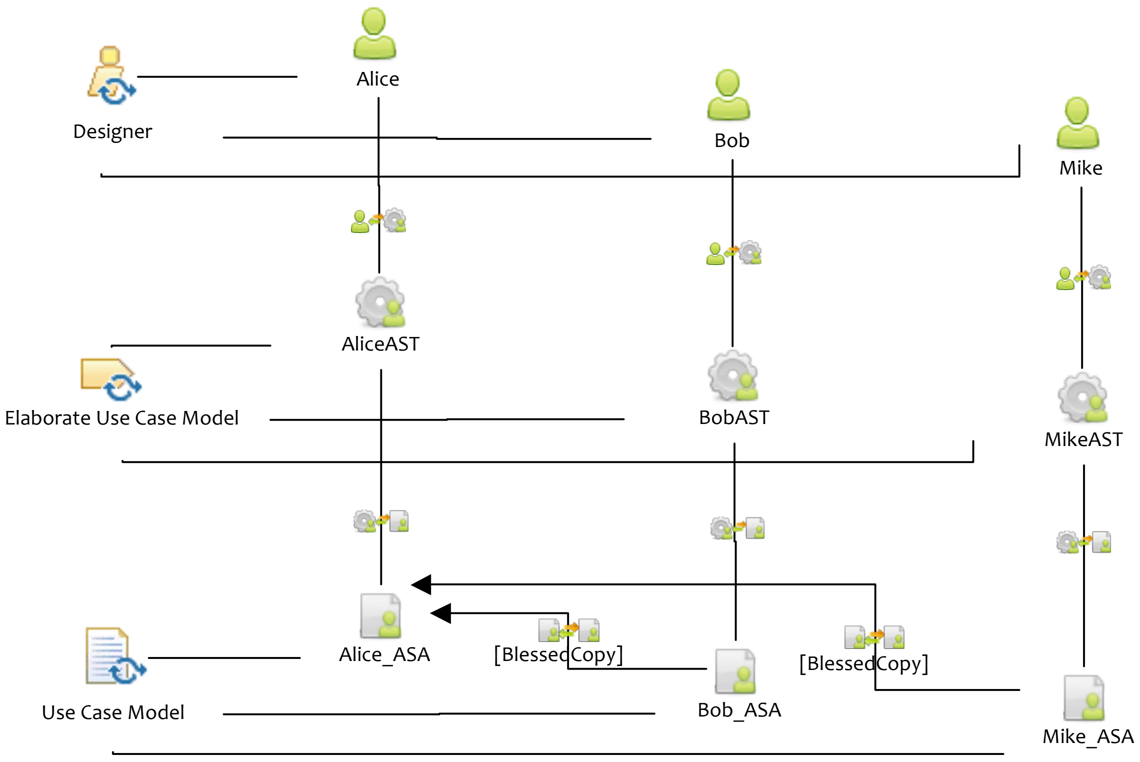

Figure 21 - Same task carried out by different actors on full-copy artifacts

3.4.2 Situation 2: The same task carried out on distinct artifacts (variation of situation 1)

As in this situation, the ActorSpecificArtifact BobASA contains only the use case Bob is working on (UC2), we rename it UC2_ASA (idem. for MikeASA which becomes UC3&4_ASA). The major change here is the graphical notation for the partial copies (UC2_ASA and UC3&4_ASA) which clearly shows they only partially represent the work product "Use Case Model".

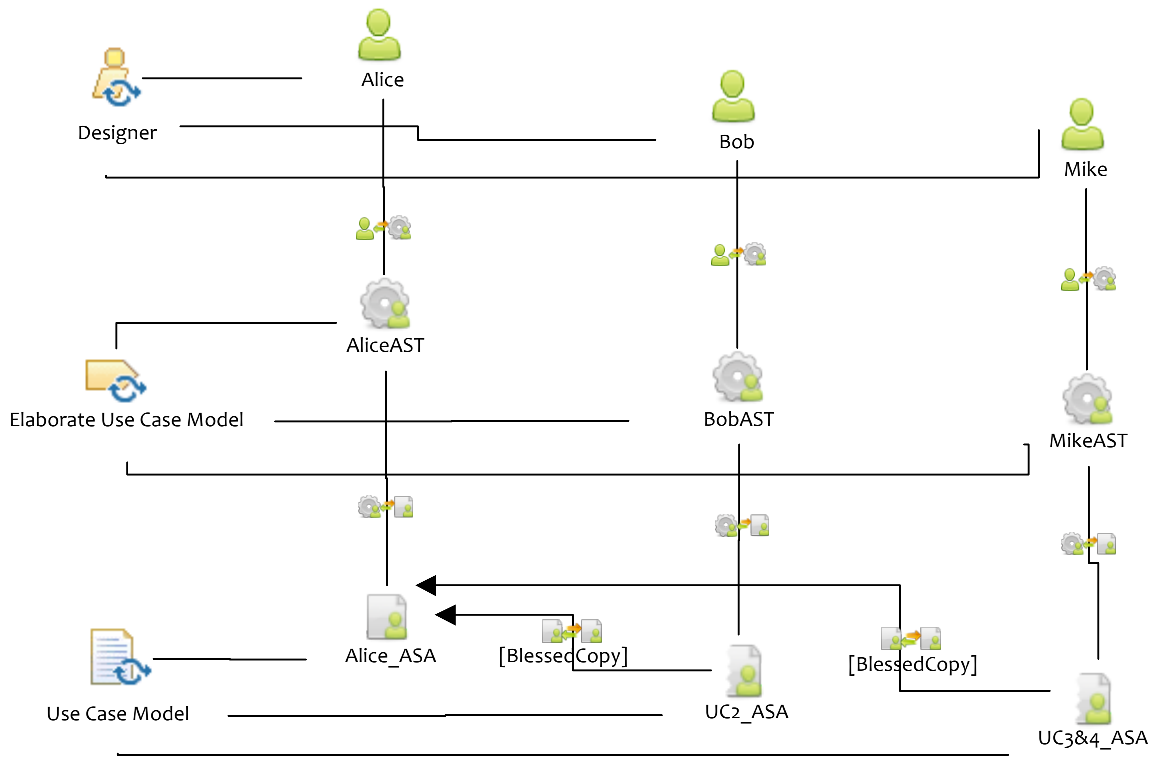

Figure 22 - Same task carried out by different actors on partial-copy artifacts

3.4.3 Situation 3: A task carried out by people playing different roles

Two different instances of ActorSpecificTask are created:

- WriteTestAST, carried out by Bob, where the tests are written, using the ActorSpecificArtifact BobTestASA
- ReviewTestAST, carried out by Tracy, where test code is reviewed for coverage, inaccuracies, and style adherence, using the ActorSpecificArtifact TracyTestASA

It should be noted that this is only one way of modeling the situation. Another process designer can chose to decompose the TaskUse “Elaborate Unit Tests” into two different TaskUse “Write Unit Test” and “Review Unit Test” (TaskUse being a WorkBreakDownElement). Then, WriteTestAST will be the only ActorSpecificTask contributing to the TaskUse “Write Unit Test”, and ReviewTestAST will be the only ActorSpecificTask contributing to the TaskUse “Review Unit Test”.

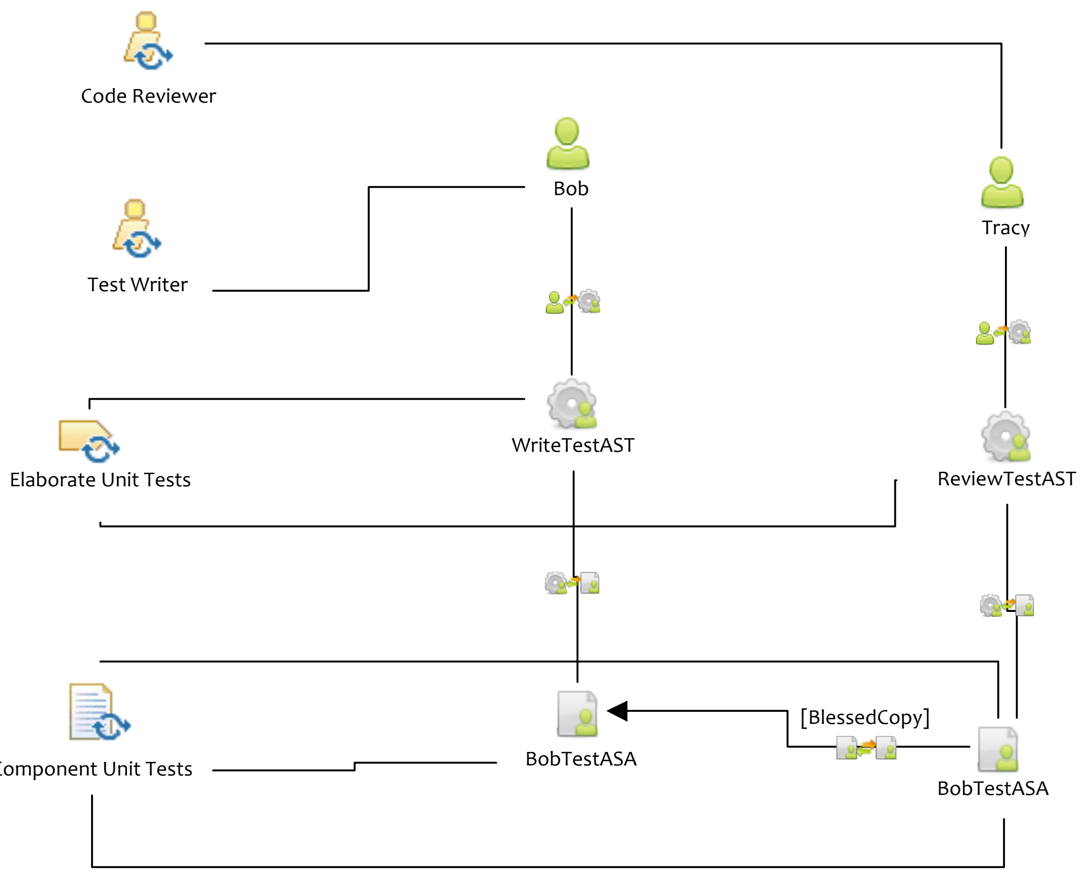

Figure 23 - A task carried out by people playing different roles

3.5 CONCLUSION

This section developed extensions to SPEM2.0 so as to allow the description of additional collaboration situations. Most of these situations arise when different people are doing the same task, or manipulate the same artifact. The modeled examples show that the extension fulfills their aim of describing these collaboration situations SPEM does not cover.

4. VIEWPOINT-ORIENTED PROCESS MODELLING

4.1 INTRODUCTION

A Galaxy process may involve a large number of participants who play different roles. Such a process may be quite complex and difficult to define. Thanks to separation of concerns, viewpoint oriented modeling is an efficient way to model complex systems. In previous work, we developed the VUML profile [Nassar et al, 2003], [Anwar, 2010] which allows to represent a system design according to functional viewpoints. In the scope of collaborative engineering, it is interesting to make an analogy between viewpoint oriented system modeling and viewpoint-oriented process modeling. Indeed, a process model can be seen as a product resulting from a dedicated process also called "meta-process".

Informally speaking, a viewpoint is the perspective, corresponding to one role (simple or composite), along which a process is modeled. Consequently, for a given process and a given viewpoint, we identify a viewpoint process that is a subset of the process containing tasks performed by the role associated to the viewpoint.

In the remainder of this section, we describe structural concepts of the CM_SPEM metamodel that related to viewpoint process modeling. More precisely, we present the CM_SPEM viewpoint structure package that contains concepts and relationships related to viewpoint process modeling.

The methodological aspect of viewpoint process modelling - that is how help process designers defining their collaborative process models in the context of Galaxy projects - will be addressed in the D2.4.2 deliverable.

4.2 THE VIEWPOINT STRUCTURE PACKAGE

4.2.1 General overview

The ViewpointStructure package is part of the CM_SPEM meta-model. It aims at grouping concepts and relationships that are related to viewpoint process modeling. It is an extension of the CM_SPEM::CollaborationStructure package made through the merge relation (see figure 15).

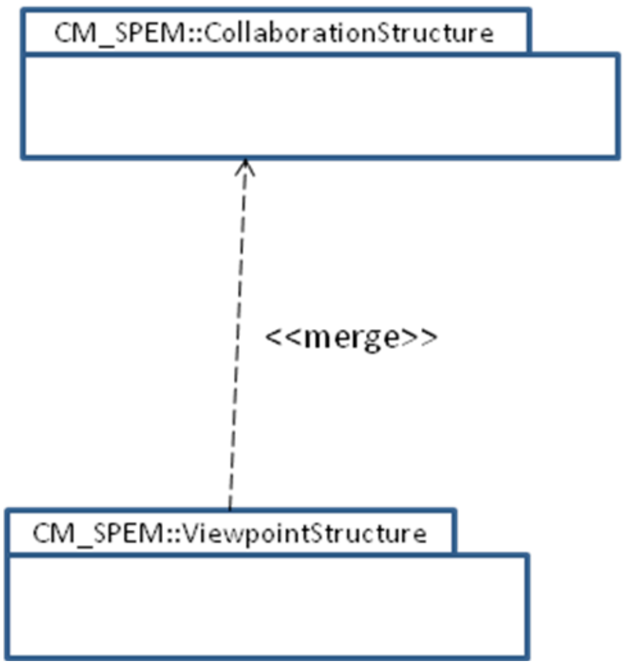

Figure 24 - Package merged by CM_SPEM::ViewpointStructure

The following figure shows the concepts of the package CM_SPEM::ViewpointStructure. For readable sake, we decided to put into this package some links to related concepts of the D2.1 deliverable. So this package contains concepts defined in the D2.1 deliverable (Galaxy, Project, Participant), concepts coming from SPEM (RoleUse, CompositeRole, TeamProfile) and CM_SPEM::CollaborationStructure (Actor). Process is redefined from SPEM::ProcessWithMethods. Viewpoint and ViewpointProcess are concepts specific of CM_SPEM::ViewpointStructure.

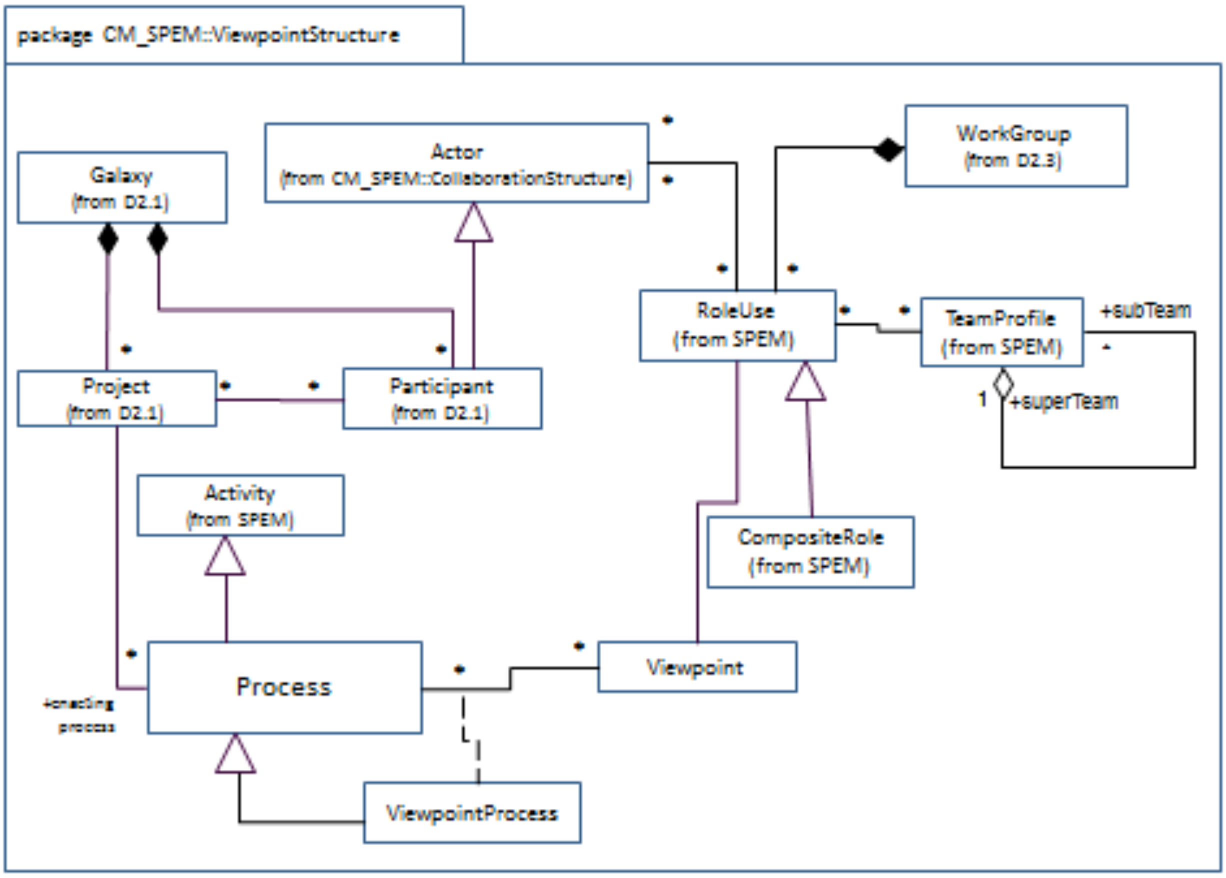

Figure 25: Package CM_SPEM::ViewpointStructure

In the following, we first recall the definition of concepts coming from the D2.1 and D2.3 deliverables and that are related to the current section, then we describe newly introduced concepts.

4.2.2 Recall: concepts coming from D2.1 deliverable

4.2.2.1 Galaxy

Galaxy represents a set of projects and participants that use the Galaxy framework to develop model-based projects in a collaborative way.

4.2.2.2 Project

In the context of Galaxy, a project is led by participants who work together to perform one or several processes.

4.2.2.3 Participant

A participant is a person who participates into a Galaxy project. It is also a human Actor, as introduced in previous section.

4.2.2.4 WorkGroup

A workgroup is a specialization for Galaxy of the concept of Group of the GCO ontology introduced in the D2.3 deliverable. It is defined as a set of actors, each actor playing one or several roles.

4.2.3 Newly introduced concepts

4.2.3.1 Process

Description

Process is redefined from SPEM::ProcessWithMethods. In the context of Galaxy, a process includes the tasks performed by participants belonging to a Galaxy project.

Association properties

- projects: Project[*]. Projects in which the process takes part.
- viewpoints: Viewpoint[*]. Viewpoints from which the process may be considered.

Semantics

A process is both a special case of Activity and a set of TaskUse corresponding to the recursive decomposition of this activity. For instance, in the context of a project using the RUP method, we can identify the (sub-)process "Elaborate the Use case Model". Such activity is decomposed into several tasks: "Identify Use cases", "Prioritize use cases", "Define Scenarios", etc. Several roles perform this process such as: "Analyst", "Designer", "Reviewer".

4.2.3.2 Viewpoint

Description

Viewpoint denotes the perspective from which a process is considered. It is associated to one RoleUse that may be single or composite.

Association properties

- role: RoleUse. The role associated to the viewpoint.
- processes: Process[*]. Processes to which the viewpoint is applied.

Semantics

A viewpoint is a way to focus on tasks of a process associated to a given role (simple or composite). For example, if we consider the process "Elaborate the Use Case Model", viewpoints may be "Analyst", "Designer", or "Reviewer" (single roles), or "Analyst/Designer" (composite role aggregating "Analyst" and "Designer" roles).

4.2.3.3 ViewPointProcess

Description

ViewpointProcess is a meta-class associated to a couple (Process, Viewpoint).

A ViewpointProcess is a process that corresponds to a process related to a given Viewpoint.

Association properties

- role: RoleUse. The role associated to the viewpoint.
- processes: Process. Processes to which the viewpoint is applied.

Semantics

A ViewpointProcess is a sub-process of a given process that corresponds to a given viewpoint. For example, considering the "Elaborate the Use case Model" process, and the viewpoint "Analyst", a ViewpointProcess will be the sub-process of "Elaborate the Use case Model" corresponding to activities of one analyst (actor having the "Analysis" role).

4.2.3.4 Examples

Let us consider the process «Elaborate the Analysis Model of a system». It is performed by participants having globally the *analysts role* (composite). Figure 25 shows the viewpointprocess corresponding to the *analysts* viewpoint.

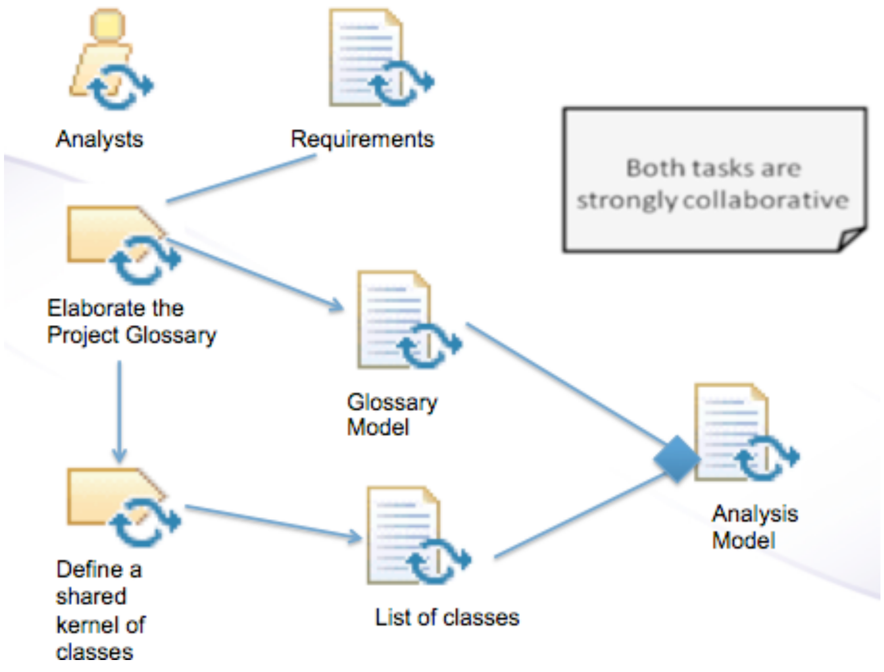
Analysts
Requirements
Both tasks are strongly collaborative
Elaborate the Project Glossary
Glossary Model
Analysis Model
Define a shared kernel of classes
List of classes

Figure 26: Analysts' viewpointprocess for elaborating the analysis model

Let us consider now the sub-process corresponding to the refinement of the task « Elaborate the Project Glossary », and let us focus on the *analyst* viewpoint (associated to a simple role). Figure 26 shows the resulting viewpointprocess.

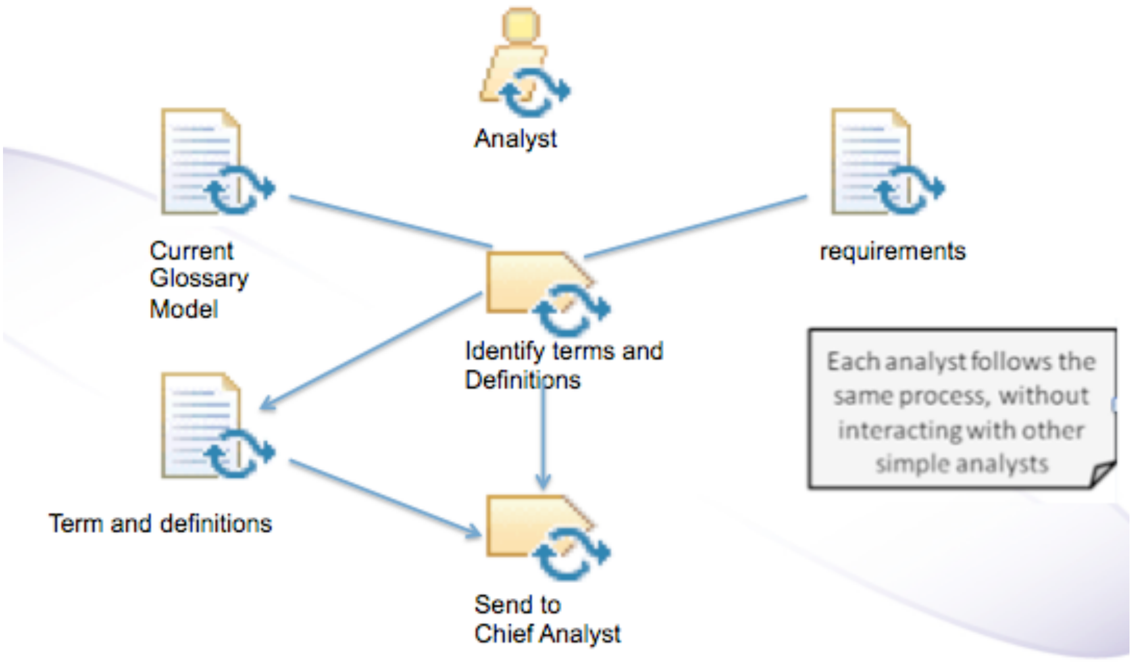
Analyst
Current Glossary Model
requirements
Identify terms and Definitions
Each analyst follows the same process, without interacting with other simple analysts
Term and definitions
Send to Chief Analyst

Figure 27: Analyst 's viewpointprocess for elaborating the glossary

If we consider now the *Chief analyst* viewpoint on the same task "Elaborate the glossary", we obtain a viewpointprocess depicted by Figure 27.

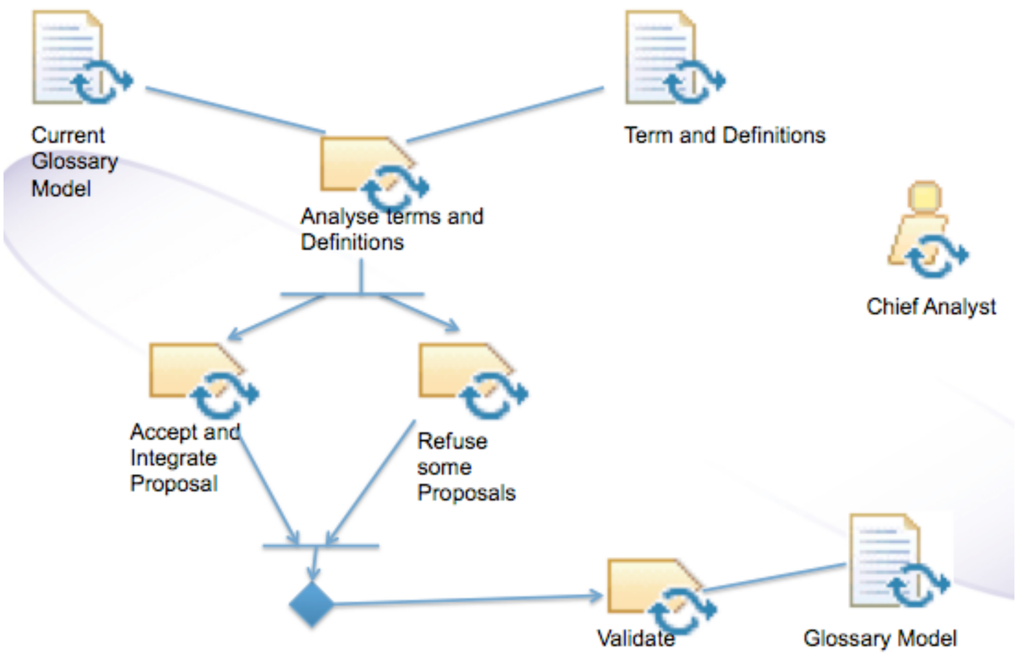

Figure 28: Chief Analyst 's viewpointprocess for elaborating the glossary

4.3 CONCLUSION

In this section we have described viewpoint-oriented process modeling. The Viewpoint concept is an efficient way to handle large and complex process models. A viewpoint is meaningful when it is associated to one or several processes. It allows describing process views of a given process, that is, tasks of the process corresponding to the role associated to the viewpoint.

As stated in the introduction, the methodological aspect of viewpoint process modeling will be addressed in the D2.4.2 deliverable. More presicely, we will discuss in D2.4.2 the best way to identify and use viewpoint processes. The result will be described as a meta-process based on strategies (mainly a top down strategy and a bottom up strategy) to follow in order to build collaborative process models in the context of Galaxy projects.

5. CONCLUSION

The work presented above in this document is part of the result of the task T2.4 of the Galaxy work package WP2. The global objective of this task is to define concepts and a methodology for modeling processes that govern model-driven collaborative development, so called Collaborative MDE Processes. The final aim is to use such process models in order to provide a computer-assisted enactment.

To achieve that goal, we have defined a metamodel – called CM_SPEM - that extends SPEM 2.0 and QVT metamodels in order to include concepts related to MDE processes. A model is defined as a specialization of a SPEM work product. Thus, activities may work on models. The concept of model transformation is defined as a specialization of SPEM work definition that has models as input/output parameters. It is also defined as a SPEM work breakdown element. Thus, model transformations may be nested by SPEM activities. A MDE process is considered as a kind of SPEM activity, which may contain activities that work on models, including model transformation, model edition, model refactoring, etc.

In this document, we have focused our work on the structural part of MDE collaborative process models. To do that, we have defined the following packages, as stated in the introduction above (see Figure 1):

- *CM_SPEM::ProcessStructure* that extends SPEM and QVT packages by redefining or adding concepts related to models transformations.
- *CM_SPEM::CollaborationStructure* that extends SPEM and other CM_SPEM packages by adding concepts related to collaborative development.
- *CM_SPEM::ViewpointStructure* that extends CM_SPEM::CollaborationStructure by adding new concepts related to viewpoint modeling.

In the D2.4.2 deliverable which is the second result of the task T2.4, we will address the enactment of MDE collaborative processes. The goal is to assist developers during a Galaxy project. To make that possible, we will enrich the CM_SPEM metamodel in order to provide process designers with concepts related to guidance and behavioral aspects.

6. REFERENCES

[Almeida, 2010]	M. A. Almeida da Silva, R. Bendraou, X. Blanc, M.P. Gervais. Early Deviation Detection in Modeling activities of MDE Processes. ACM IEE International Conference on Model Driven Engineering Languages and Systems MODELS 2010, Oslo, Norway
[Anwar, 2010]	Anwar A., S., Coulette B., Nassar M., Kriouile A,. (2010). A Rule-Driven Approach for composing Viewpoint-oriented Models. Journal of Object Technology, ETH Swiss Federal Institute of Technology, Vol. 9 N. 2, pp. 89-114.
[Bézivin, 2004]	Bézivin, J., Breton, E.: Applying the Basic Principles of Model Engineering to the Field of Process Engineering. The European Journal for the Informatics Professional. 5, 27-33 (2004)
[Kabbaj, 2008]	M. Kabbaj, R. Lbath, B. Coulette. “A Deviation Management System for Handling Software Process Enactement Evolution”. In International Conference on Software Process ICSP 2008
[Marcaillou, 1994]	Marcaillou, S., Coulette, B., Kriouile, A., Visibility: a new relationship for complex system modelling, International Conference TOOLS USA’94, Santa Barbara, Californie, USA, 1994.
[Nassar et al, 2003]	Nassar, M., Coulette, B., Crégut, X., Marcaillou, S and Kriouile., A. “Towards a View based Unified Modeling Language”. In ICEIS’03, Angers, France, 2003.
[OMG, 2008-a]	OMG SPEM2.0, “Software & System Process Engineering Metamodel”, OMG document, final adopted specification, ptc/07-03-03, at http://www.omg.org
[OMG, 2008-b]	Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, at http://www.omg.org/spec/QVT/1.0/PDF/