Software Process Engineering Management

The Unified Process Model (UPM)

Initial Submission

OMG document number ad / 2000-05-05

May 12, 2000

Submitted by

IBM
Rational Software
SofTeam
Unisys
Nihon Unisys Ltd.
Alcatel
Q-Labs (ex-Objectif Technologies)

Supported by

Valtech
Toshiba

Table of Contents

1 Introduction

The following companies are pleased to submit this specification in response to the Software Process Engineering (SPE) Management RFP
(Document ad/99-11-04):

• IBM Corporation, Steve Cook	scook@acm.org
• Rational Software, Philippe Kruchten	pbk@rational.com
• SofTeam, Philippe Desfray	philippe.desfray@softeam.fr
• Unisys, Sridhar Iyengar	Sridhar.Iyengar2@unisys.com
• Nihon Unisys Ltd, Hiromichi Iwata	Hiromichi.Iwata@unisys.co.jp
• Alcatel, Laurent Rioux	Laurent.Rioux@alcatel.fr
• Q-Labs, Annie Kunzmann-Combelles	akc@objectif.fr

We also acknowledge support from:

• Valtech, Craig Larman	craig.larman@valtech.com
• Toshiba, Mari Natori	marin@sitc.toshiba.co.jp

1.1 Overview

This document presents the *Unified Process Model* (UPM). This model is used to describe a concrete software development processes or a family of related software development process. Process enactment is outside the scope of UPM, although some examples of enactment are included for explanatory purposes.

1.2 Modeling Approach

We take an object-oriented approach to modeling a family of related software processes and we use the UML as a notation. Figure 1 shows the four -layered architecture of modeling as defined by the OMG. A performing process—that is, the real-world production process—as it is enacted, is at level M0. The definition of the corresponding process is at level M1. For example, the Rational Unified Process 2000 (RUP2000) or the IBM SI Method is defined at level M1. Both a generic process like RUP and a specific customization of this process used by a given project, are at level M1. We focus here on the metamodel, which stands at level M2 and serves as a template for level M1.

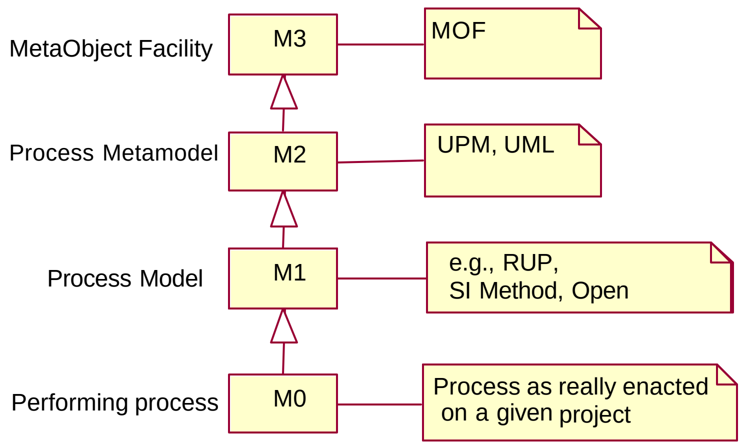


Figure 1—Levels of modeling

1.3 Scope

The UPM is a metamodel for defining processes and their components. A tool based on UPM would be a tool for process authoring and customizing. The actual enactment of processes—that is, planning and executing a project using a process described with UPM, is not in the scope of this model. See Section **11** for further explanation of the relationship between the UPM and an actual process enactment.

In this proposal, we are limiting ourselves to defining the minimal set of process modeling elements necessary to describe any software development process, without adding specific models or constraints for any specific area or discipline, such as project management or analysis.

We believe this is the appropriate approach for the software process engineering domain, and any attempt to standardize a more complex and detailed model at this time would be both unwise and ineffective. The standard wants to *accommodate* a large range of existing and described software development processes, and not *exclude* them by having too many features or constraints.

1.4 Terminology

There are a large number of process models and standards. Each one uses slightly different terminology, sometimes with different meaning for the same English word or phrase. For example, a `phase' in Fusion [13] is called a `core workflow' in the Rational Unified Process (RUP) [1] and a `domain' in IBM's SI Method. We will designate it as a `discipline' here. OPEN [4] and the Rational Unified Process [1] both use the word `activity' but with a different meaning. We have provided "translations" (aliases or synonyms) to help in understanding. This also allows the naming of various process elements by the appropriate term in various languages: Japanese, French, and so on. See Annex 1 for a comparison table and Section 14 for the Glossary.

1.5 Relationships to Other OMG Specifications

UML

The Unified Modeling Language (UML) is a graphical language for modeling discrete systems. Although the UML is not necessarily tied to any particular application area or modeling process, its greatest applicability is in the area of object-oriented software design. Version 1.1 of the UML was submitted to the Object Management Group in September 1997 in response to an OMG RFP requesting a standard approach to object-oriented modeling. The proposal was ratified by the OMG in November 1997. Version 1.3 of the UML was finalized in June 1999 and is the version referred to throughout this document.

The UML is defined by a metamodel, which is itself defined using a subset of UML that maps onto the MOF (Meta-Object Facility). The UPM metamodel is defined similarly.

The purpose of the Unified Process Model (UPM) is to support the definition of software development processes specifically including those processes that involve or mandate the use of UML, such as the Rational Unified Process.

UML Profile

A UML profile is a variant of UML that uses the extension mechanisms of UML in a standardized way, for a particular purpose. Currently there is no normative definition of a UML profile, however, the Business Object Initiative RFPs gave the following working definition of a UML profile (OMG document ad/99-03-10).

A UML profile is a specification that does one or more of the following:

- Identifies a subset of the UML metamodel (which may be the entire UML metamodel).

- Specifies "well-formedness rules" beyond those specified by the identified subset of the UML metamodel. "Well-formedness rule" is a term used in the normative UML metamodel specification to describe a set of constraints written in natural language and UML's Object Constraint Language (OCL) that contributes to the definition of a metamodel element.

- Specifies "standard elements" beyond those specified by the identified subset of the UML metamodel. "Standard element" is a term used in the UML metamodel specification to describe a standard instance of a UML stereotype, tagged value or constraint.

- Specifies semantics, expressed in natural language, beyond those specified by the identified subset of the UML metamodel.

- Specifies common model elements (that is, instances of UML constructs), expressed in terms of the profile.

A green paper (OMG document ad/99-12-32) has been issued that proposes a set of requirements for a more formal specification of UML profiles. This document is intended as input to the definition of UML 1.4.

The UPM metamodel is not defined as a UML profile, primarily because it is not at all clear how UPM can be sensibly mapped onto a subset of the UML metamodel. Section 13 of this proposal discusses why this is so.

The UPM metamodel includes a class that reifies the use of a UML profile within a process and is discussed further in Section 8.

MOF 1.3 and XMI

The Meta-Object Facility (MOF) is the OMG's adopted technology for defining metadata and representing it as CORBA objects. The MOF 1.3 specification was finalized in September 1999 (OMG document ad/99-09-05). A MOF metamodel defines the abstract syntax of the metadata in the MOF representation of a model. The MOF model itself describes the abstract syntax for representing MOF metamodels. MOF metamodels can be represented using a subset of UML syntax. UPM is presented as a MOF metamodel.

XMI (XML Metadata Interchange) is the OMG's adopted technology for interchanging models in a serialized form (OMG document ad/98-10-05). XMI version 1.1 was formally adopted by the OMG in February 2000 (OMG document ad/99-10-04). XMI focuses on the interchange of MOF metadata; that is, metadata conforming to a MOF metamodel.

XMI is based on the W3C's eXtensible Markup Language (XML) and has two major components:

- The XML DTD Production Rules for producing XML Document Type Definitions (DTDs) for XMI encoded metadata. XMI DTDs serve as syntax specifications for XML documents, and allow generic XML tools to be used to compose and validate XMI documents.

- The XML Document Production Rules for encoding metadata into an XML compatible format. The production rules can be applied in reverse to decode XMI documents and reconstruct the metadata.

Because UPM is defined as a MOF metamodel, XMI can be used:

- to transform the UPM metamodel into a UPM Document Type Definition
- to transfer process models based on UPM as XML documents, based on the UPM DTD
- to transform the UPM metamodel itself into an XML document, based on the MOF DTD, for interchange between MOF-compliant repositories

Workflow

Within the OMG there are three initiatives that come under this heading.

The first is the Joint Workflow Management Facility (OMG document bom/99-03-01). The scope of this facility is workflow enactment and it supports Workflow Client Applications, Interoperability, and Process Monitoring as described in the Workflow Reference Model. None of these areas overlaps the SPE submission, which addresses the domain of process description, not process enactment.

The second is the Workflow Resource Assignment Interfaces RFP (OMG document bom/2000-01-03), which asks for submissions to extend the capabilities of the adopted workflow management specification in the areas of the assignment and selection of resources. The scope of this facility is also process enactment and so does not overlap the SPE submission.

The third area of interest is Process Definition. At this time no request for proposals has been issued. The matter is still under consideration, pending discussions within the UML RTF and the UML 2.0 working group about how UML Activity Diagrams will be supported and/or extended. This discussion overlaps the scope of the current submission. We assume in this submission that an appropriate resolution of the use of Activity Diagrams to describe workflows will occur, and we provide a class in the proposed metamodel intended to represent an element on a workflow diagram. Furthermore, in our experience, although activity diagrams can be useful to represent flows through a software development process, they are not usually the most effective way to describe sequencing development activities. This subject is addressed in more detail in Section 10.

1.6 Proof of Concept

The (meta)model presented here supports both the Rational Unified Process and IBM's SI Method. Examples throughout the text show how particular elements in the model are used in these and other processes. It is also supported by the Rational Process Workbench (RPW), which is a process authoring tool based on UML.

2 Mapping to RFP Requirements

2.1 Mandatory Requirements:

Four-layer Architecture

- *Submissions shall conform to the four-layer architecture defined by the OMG.*

UPM sits at level M2 in the four-layer architecture and further details are found in Sections 1.2 and 13.

Relationship to UML and MOF

- *Metamodels shall be clearly positioned in relation to the UML metamodel and built using the MOF meta-metamodel. Relationships between these metamodels shall be identified and specified.*

The UPM metamodel is defined using a subset of UML in a similar way to UML and to MOF. This subset of UML corresponds to the facilities supported by MOF. The UPM metamodel is largely independent of the UML metamodel, with the exception of the use of Activity Diagrams. UPM has not been defined as a UML profile for reasons discussed in Section 13.

XMI DTD

- *A submission shall include an XMI DTD for a submitted metamodel.*

Such a DTD is not included in this initial submission. It will be included in the final submission.

Basic Concepts

- *The metamodel shall address at least the following concepts: Tasks, Techniques, Roles, Products, Phases. Responses are not required to use these exact names.*

The UPM supports the description of these concepts, not their enactment, in the following ways:

- Tasks are modeled by ActivityKind (see Section 6.3)
- Techniques are modeled by Technique (see Section 8)
- Roles are modeled by RoleKind (see Section 6.4)
- Products are modeled by ArtifactKind (see Section 6.1)
- Phases are modeled by Phase (see Section 10.3)

Process Examples

- *Submissions shall submit two or more examples of processes that use the submitted metamodel.*

UPM is specifically designed to underpin the Rational Unified Process and IBM's family of methods, including the SI Method deployed throughout IBM Global Services.

Process Patterns and/or Components

- *Submissions are required to define constructs that enable the creation and use of reusable process patterns and/or components.*

The construct ProcessComponent, described in more detail in Section 9.1, represents such a reusable piece of process.

Glossary

- *Submissions shall include a full glossary of SPE terms. These terms shall have a clearly-defined relationship to the constructs defined in the submitted SPE metamodel.*

Section 14 of this document provides a glossary of the main terms used in the metamodel.

Support for UML

- *The facility shall support the use of UML for software engineering modeling and process modeling. A specification of relationships between SPE constructs and UML constructs is required, wherever such relationships exist. Facilities providing help in UML usage, depending on the activity and on the development context, shall also be defined.*

The UPM can be used to define all kinds of processes, including those focused on the specific use of UML. Instances of Guidance subclasses for describing UML practices and tools would be created for an UML-specific process.

Categories

- *A submission shall provide the facility to define a standardized set of categories. A submission shall provide the ability to classify all process elements using these categories.*

The meta-class Discipline, explained in Section 9.3, supports a categorization of process elements based on a partitioning of the ActivityKinds.

Natural Language Translation

- *A submission shall be organized so that a process can readily be translated between different natural languages without losing its structure.*

Natural languages are represented by the class Language and are described further in Section 5. The visible name of a process element in a given natural language is separated from the internal model name. All structured textual elements are kept separate from the structure of the process itself in the class TextualDescription, which is also associated to one Language.

Graphical Notation

- *Responses shall include graphical notations or default to UML notations. Where a response makes notation recommendations other than UML it shall show the relationship between those recommendations and other established process modeling notations; for example, IDEF0. If a recommended notation is not UML-based, responses shall explain why a different notation is better.*

The intent of the UPM is to offer graphical notations for depicting software engineering processes that are similar to those of UML. However, just using UML diagrams is not readily feasible because of the difficulties in making UPM a UML profile as discussed in Section 13. If we fail to make the UPM a UML profile, then some mapping of concepts between UPM and UML will be introduced in the final submission to show the correspondence between UPM entities and UML.

Ideally a graphical notation for UPM would use class diagrams to depict dependencies between process elements, such as work-breakdown structure or product structure. It would use activity diagrams to depict sequencing of activities or collaboration diagrams to show interaction between various roles.

Special icons need to be used to denote process-related concepts—artifacts, roles, activities, and so on—to make these diagrams more expressive.

Annex 3 of this document provides an example of what a process notation could look like.

Compared to IDEF0, which focuses mostly on activities, their decomposition, and their sequencing, a UML-like set of process diagrams gives a much wider palette of expression and allows the use of existing UML-supporting tools to model the process.

2.2 Optional Requirements

Submission as a UML Profile

- *A submission may define a UML profile.*

This submission is not presented as a UML profile. Section 13 discusses the reasons for this.

Definition of Process Patterns and/or Components

- *A submission may define actual process patterns and/or components.*

Actual process components are not included in the UPM.

Reification of UML Profile Concept

- *Submissions may reify the UML profile concept. The way in which profiles may constrain the development process, notations or tools may be emphasized. Relationships between profiles and activities and between profiles and work products may be clarified.*

The class UMLProfile, described in Section 8, meets this requirement.

3 Conceptual Model

At the core of the Unified Process Model (UPM) is the idea that a software development process is a collaboration between abstract active entities called *roles* that perform operations called *activities* on concrete, tangible entities called *artifacts* [20]. Figure **2** depicts this fundamental conceptual model using the UML notation for a class. Figures 2 and 3 are not part of the proposed model and are given solely for explanatory reasons. They are intentionally very incomplete.

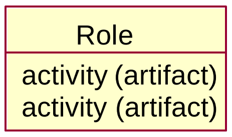


Figure 2—Conceptual model

Multiple roles interact or collaborate by exchanging artifacts and triggering the execution, or enactment, of certain activities. The overall goal of a process is to bring a set of artifacts to a well-defined state.

From this model, a first step consists of "reifying" role, activity, and artifact. This leads to the simple model shown in Figure **3**.

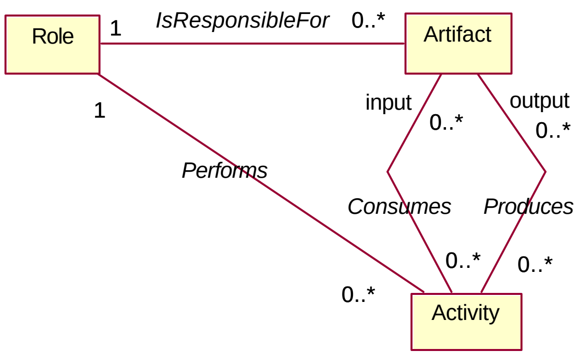


Figure 3—Reifying the conceptual model: roles, artifacts, and activities

Note that this simple conceptual model describes what happens during process enactment and, as such, is not part of the formal model defined by this submission. We return to this model in Section 11
where the relationship between described processes and their enactment is explained in more detail.

4 Package Structure

The UPM metamodel is divided into six packages, shown in Figure **4**, called Names, BasicElements, Process Structure, Process Components, Guidance and Process Lifecycle. We address each package in turn in the next six sections.

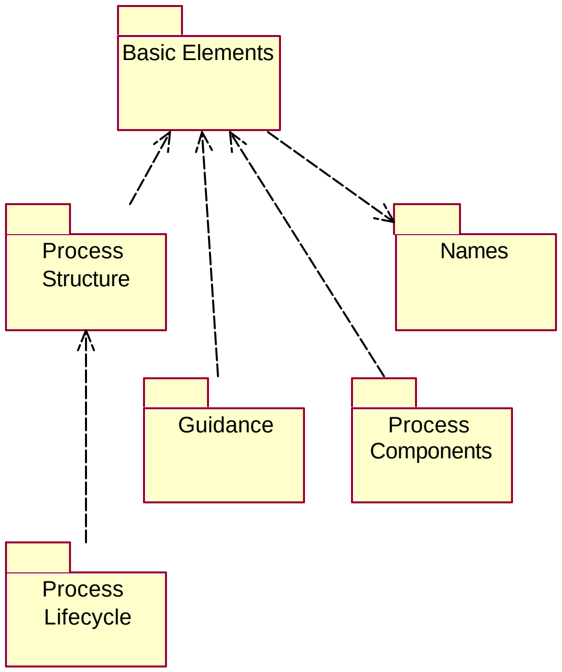


Figure 4—UPM packages

5 Basic Elements

This package, detailed in Figure **5**, defines the basic elements from which the rest of the model is derived.

5.1 NamedElement, VisibleName, and Language

Most elements of the process metamodel are a specialization of a common abstract class, called NamedElement, with a single attribute *internalName.* This permits elements to have a textual name by which they can be referred in the process model.

To allow the expression of a process in any natural language, a named element can be associated to any number of VisibleName, which contains a Unicode string of the name of the element in a natural language. An association to the class Language specifies the language.

5.2 ProcessDefinitionElement and TextualDescription

Most elements of the process metamodel are also a specialization of a common abstract class, called *ProcessDefinitionElement*, introduced to capture common attributes such as a textual description. In particular, all process elements have one or several descriptions in natural languages. WorkDefinition, ArtifactKind, RoleKind, and Guidance are the main subclasses of ProcessDefinitionElement.

Relationships

- TextualDescription: one or more textual descriptions in natural language are associated with each instance of a process definition element for its static description. This can take into account variants of process, such as linguistic variants, or more or less complex processes or techniques. For some ProcessDefinitionElements, this textual description could be structured; for example, by purpose, by properties, and so forth. A relation to the class Language specifies the language of the description. The format of the description is specified by an attribute format, such as HTML, PDF, Microsoft Word, and so on.

- ▪ Guidance: one or more guidance elements are associated with a process element to provide help to the practitioner. Guidance is explained in Section 8.

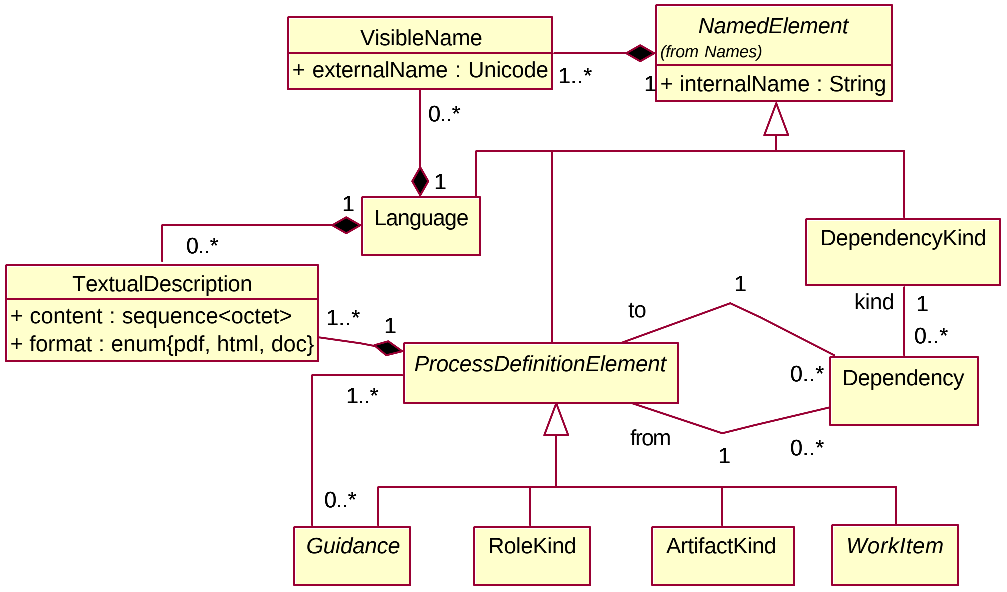


Figure 5—Basic Elements package

5.3 Dependency and DependencyKind

The definition of a process may introduce dependencies or relationships between process definition elements, such as traceability dependency. This is the responsibility of the *Dependency* class, which represents a one-way dependency between process definition elements, and the *DependencyKind* class, which represents the kind of dependency.

For example, an important document in IBM's SI Method is the Work Product Dependency diagram, represented in Figure **6**. The rectangles in this diagram indicate Work Product Descriptions—in UPM terms, instances of ArtifactKind as described in Section 6.1. The arrows represent instances of Dependency associated with an instance of DependencyKind named Work Product Dependency.

Another DependencyKind is the Requires dependency used to create process families (see Section 9.2). It indicates that the presence of a process definition element in a process or family of processes explicitly requires the presence of another one, even in the absence of model relationship.

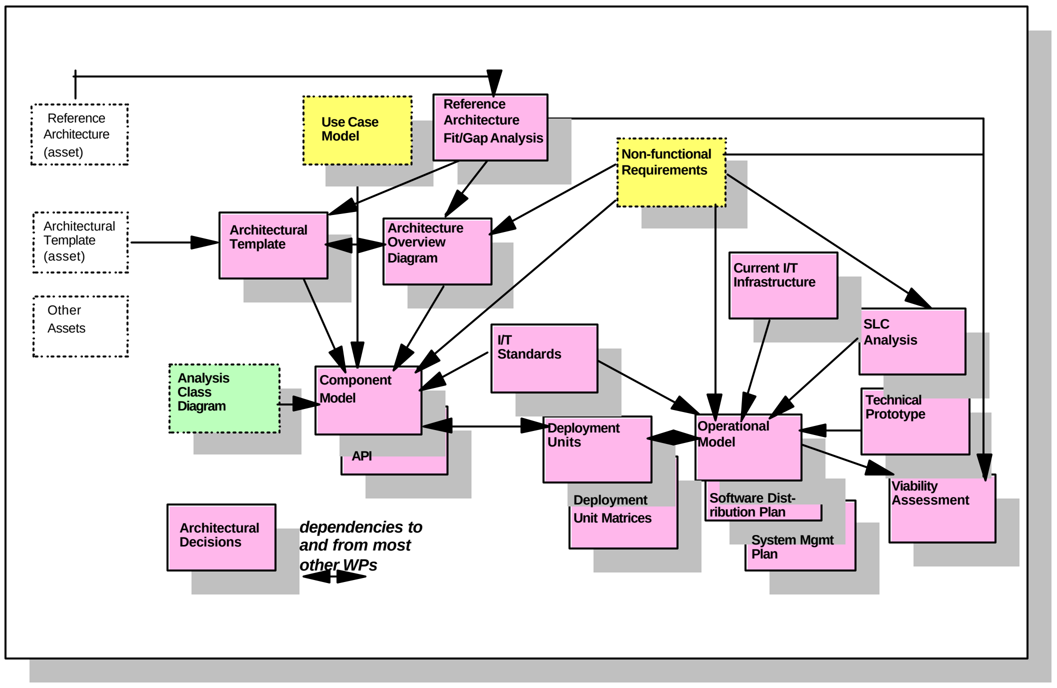


Figure 6—A Work Product (Artifact) Dependency diagram from IBM's SI Method

6 Process Structure

This package, shown in Figure **7**, defines the main structural elements from which a process description is constructed.

6.1 ArtifactKind and ArtifactName

An *Artifact* is anything produced, consumed or modified by a process. It may be a piece of information, a document, a model, source code, and so on. An *ArtifactKind* describes one kind of artifact.

Relationships

- ArtifactKind is a specialization of ProcessDefinitionElement (from Basic Elements).
- ArtifactKinds can be composed of other ArtifactKinds (aggregate artifacts), using the helper class ArtifactName.
- ArtifactKinds can be defined as individually-named inputs and outputs of a WorkDefinition, through the class ArtifactUsageName, which indicates the ArtifactKinds it uses.
- The attribute hasWorkPerArtifact indicates that you need multiple instances of the WorkDefinition—one per instance of the corresponding artifact. For example, *Write the code of a class* may have *Coding standards* and *Class* as inputs but it is replicated once per class, not per coding standard.

Instances of *ArtifactName* designate the names of artifacts composed within the scope of an *ArtifactKind.* With this class, an artifact kind can be named differently within the scope of different containing artifact kinds; for example, a Plan could be called “Risk Plan” in one scope and “Result Plan” in another. A given *ArtifactKind* may be multiply-contained within the same scope under different names; for example, a composite deliverable containing several Class Diagrams could contain an Analysis Class Diagram and a Design Class Diagram.

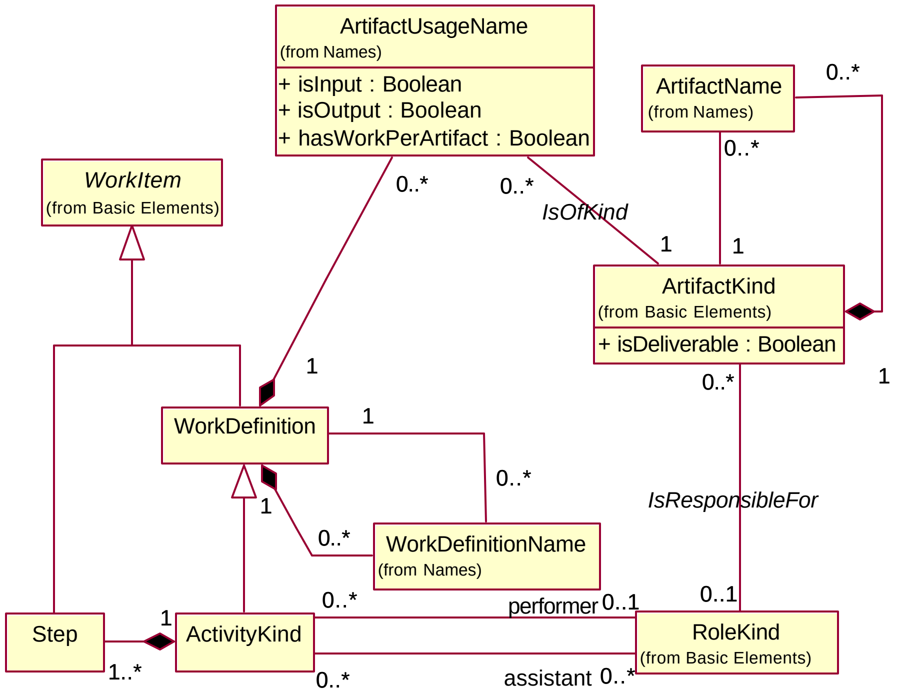


Figure 7— Process Structure package

The *isDeliverable* attribute on ArtifactKind is true if that artifact is defined as a formal deliverable of the process.

Examples

”Design Model” is an ArtifactKind that describes design models, which are artifacts. “Software development plan” is an ArtifactKind that is an aggregate of several other ArtifactKinds, such as documents and plans, designated by name; for example, “Risk Plan”.

Synonyms

‘Artifact’ is the term used in the RUP for the description of the artifact; the IBM process uses the term ‘Work Product Description’. Other processes use the terms ‘deliverable’ or ‘product’.

Note

Deliverable is not a major element in UPM because not all artifacts are deliverable, and whether an artifact is delivered or not may change during the enactment.

6.2 WorkItem, WorkDefinition, and WorkDefinitionName

A *WorkItem* is a ProcessDefinitionElement that describes the work performed by roles. It is an abstract class, with two subclasses: WorkDefinition and Step. WorkItems can be used in activity diagrams.

WorkDefinition is a ProcessDefinitionElement, a subclass of WorkItem that describes the work performed by roles. Its main subclasses are ActivityKind, as well as Phase, Iteration, and Lifecycle (in the Process Lifecycle package). Unlike WorkItem, WorkDefinition can be recursively structured, and has explicit inputs and outputs.

A *WorkDefinitionName* is a helper class whose instances designate work definitions when creating aggregate work definitions. The introduction of this helper class permits a WorkDefinition to be referred to with multiple names, similarly to the use of ArtifactName for Artifacts.

Relationships

- WorkDefinitions can be composed of other WorkDefinitions (aggregate Work Definitions), using the helper class WorkDefinitionName.
- A WorkDefinition is related to the ArtifactKind it uses through the ArtifactUsageName class, which specifies whether they are used as input or output. The work described in the WorkDefinition uses the input artifacts, and creates or updates the output artifacts. Through the name attribute of the ArtifactUsageName class, a given ArtifactKind can form multiple, differently named inputs and/or outputs of a WorkDefinition.

The familiar concept of Work-Breakdown Structure (WBS) can be described using two UPM constructs:

1. Composition using WorkDefinitionName provides the means to describe that one ActivityKind is composed of another and, therefore the hierarchical nature of the WBS.
2. The Dependency concept between ProcessDefinitionElements (inherited by WorkDefinition) provides the ability to sequence between elements of the WBS at the same level or to describe other dependencies between ActivityKinds, ActivityGroups, and so forth.

6.3 ActivityKind and Steps

ActivityKind is the main concrete subclass of WorkDefinition. It describes a piece of work performed by one role: the tasks, operations, and actions that are performed by a role or with which the role may assist. An ActivityKind can be decomposed into atomic elements called *Steps*.

Steps are a subclass of WorkItem and, therefore, cannot be decomposed. Steps are performed by the same role as the enclosing activity.

Relationships

- ActivityKind inherits input and output artifacts.
- An ActivityKind refers to a RoleKind that is the performer of the described activity and may refer to additional RoleKinds that are the assistants in the activity.
- An ActivityKind does not use the composition structure inherited from WorkDefinition; instead composition within ActivityKind is done using Steps. Steps inherit the context of the enclosing ActivityKind in terms of the RoleKind and ArtifactKinds they use.

Examples

In the RUP, *Find use case and actors* is an example of ActivityKind. It is decomposed in half a dozen "steps" in the RUP: *Find actors, …., Check the results.*

In IBM's SI Method, the "activity" *Specify Solution Requirements* is an example of a WorkDefinition. It is decomposed into several "tasks", modeled by UPM's ActivityKind, such as *Detail Usability Requirements.*

Synonyms

The Rational Unified Process uses 'activity' composed of a partially ordered set of 'steps'. The IBM process defines 'activities' that corresponds to UPM WorkDefinition consisting of 'tasks' and 'subtasks' that corresponds to UPM ActivityKinds.

6.4 RoleKind

A *RoleKind* defines a role, possibly a composite role, which a person, or a group of people, may be called upon to play in a process. RoleKind defines responsibilities over specific artifacts, and defines the roles that perform and assist in specific activities.

Relationships

- A RoleKind is a specialization of ProcessDefinitionElement.
- A RoleKind is responsible for a set of artifacts.

- A RoleKind is normally the performer and / or assistant for several ActivityKinds.

Synonyms

This concept is called ‘role’ in the IBM SI method and in OPEN [4], and ‘worker’ in the Rational Unified Process [1, 3]. We have also encountered ‘agent’.

Examples

In the Rational Unified Process, examples of workers are *Architect*, *Analyst*, *Technical Writer*, and *Project Manager* to name a few.

Notes

A RoleKind is not a person. A given person may be acting in several roles and several persons may act as a given role.

7 Names

The Names package, illustrated in Figure **8**, contains just the classes that represent the names that process description elements have in various contexts. The function that each of these classes has in the metamodel has been described in the Process Structure package. The purpose of the Names package is just to show that each of the Name classes inherits from NamedElement and, in particular, has an internalName attribute.

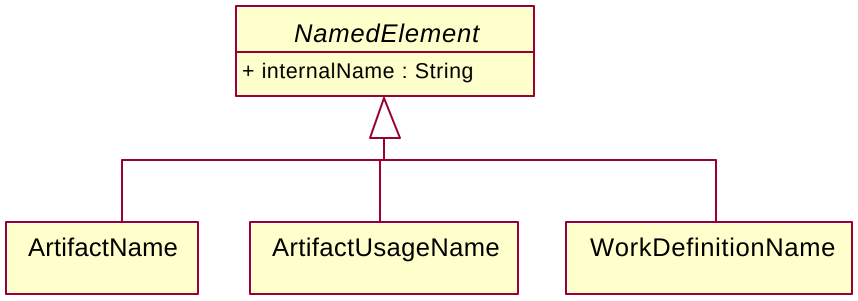


Figure 8—Names package

8 Guidance

The Guidance package, shown in Figure **9**, includes classes representing various kinds of support to assist the user of a process description.

The *Guidance* package represents any supporting guidance, including instructions, procedures, technique, guidelines, checklists, examples, templates, tool guides, metrics, and standards that may be required to help the practitioner accomplish what is described in a ProcessDefinitionElement. Guidance itself is an abstract class, and its association with ProcessDefinitionElement was shown in the Basic Elements package.

Tools are a resource needed to run a process. Tools may be associated with specific activities, which they may completely automate.

Example

In the RUP, tools are related to activities and artifacts by ToolMentors, which are specializations of Guidance.

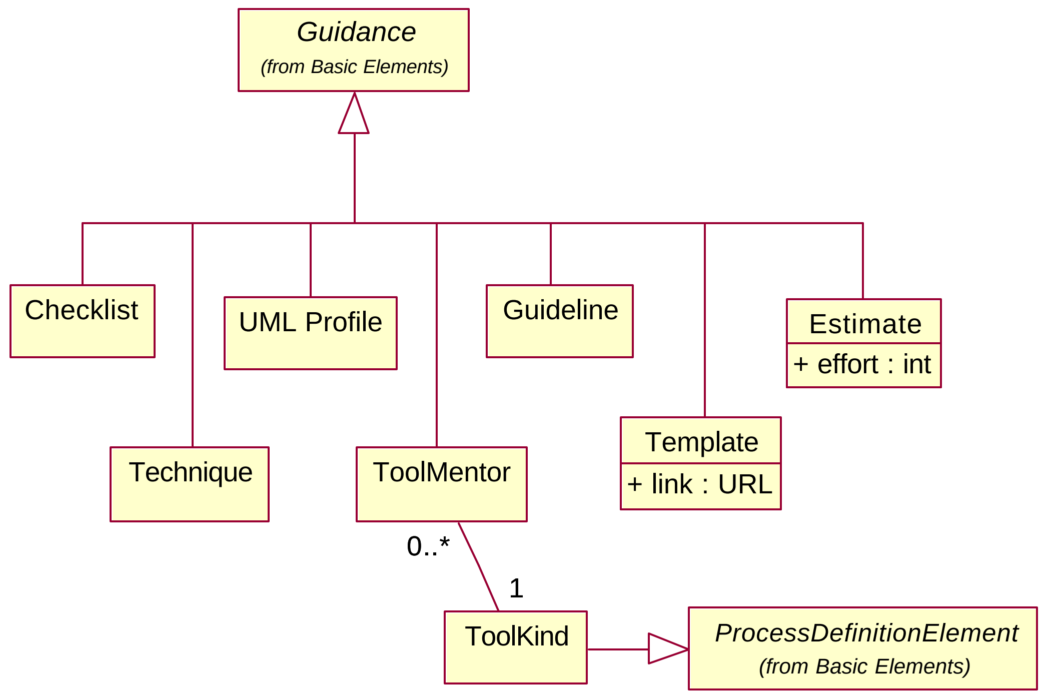


Figure 9—Guidance package

Relationships

- Guidance is a specialization of ProcessDefinitionElement.
- Each Guidance may be associated with one or more ProcessDefinitionElements and many Guidances may be associated with each ProcessDefinitionElement. Some specific uses of this flexible association are explained in the descriptions of different kinds of Guidance described below.

Synonym

The OPEN process uses the term ‘technique’. Other processes use ‘procedure’ or ‘directive’.

Checklist is a subclass of Guidance. A checklist is a document representing a list of elements that need to be completed.

Technique is a subclass of Guidance. A Technique is a detailed, precise “algorithm” used to create an artifact.

Example of a Technique:

Using Petri nets, interviewing, identifying relevant classes, eliminating excessive inheritance, constructing a GANTT chart, and tracking progress with the earned-value method

UMLProfile is a subclass of Guidance. A UML profile provides mechanisms that specialize UML for a specific target such as C++, Java, and CORBA or for a specific purpose such as analysis, design, and so on. Every development activity using UML can be ruled by a profile that dictates those UML consistency rules that need to be applied or which UML model element is relevant for the current context and focus of the activity.

The exact representation of a UML profile in the metamodel needs to be aligned with the developments in UML 1.4 and UML 2.0.

Examples of a UML Profile

“UML for EJB”, “UML for Analysis”, “UML for CORBA”

Figure 10 presents a diagram example of such
an approach, where activities are connected to UML profiles.

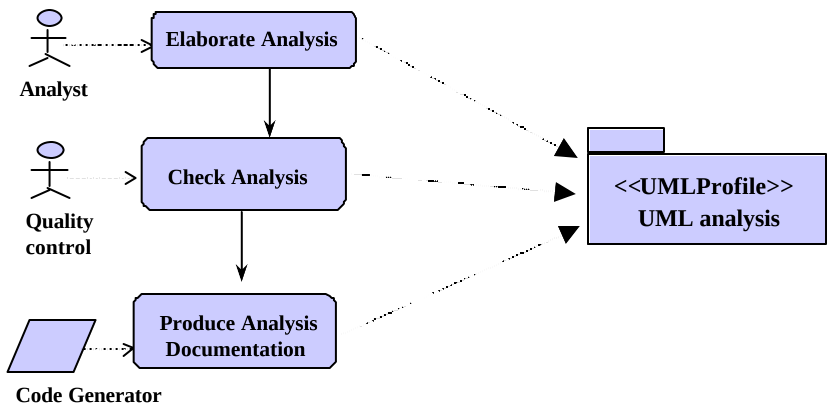


Figure 10—Example of a process connecting activities to UML profiles

ToolMentor is a subclass of Guidance. A ToolMentor shows how to use a specific tool to accomplish an activity. Each ToolMentor is associated with a single ToolKind and inherits the association with the ActivityKind it supports from Guidance.

Example of a ToolMentor

“Using Rational ClearCase to Check Out and Check In Configuration Items”

ToolKind represents a particular kind of tool to be used in the process.

Examples of a ToolKind:

A CASE tool such as Rational Rose, a programming tool such as VisualAge for Java, a testing tool, and so on.

Guideline is a subclass of Guidance. A Guideline is a set of rules and recommendations on how a given artifact must look or must be organized.

Example of a Guideline:

In the Rational Unified Process, the *Java Programming Guidelines* are guidance used in the implementation of a design class, as well as input for the activity of code review.

In this example, we see connections from RoleKind occurrences such as “Analyst” as performers, to ActivityKind occurrences such as “Elaborate Analysis”, and from ActivityKind occurrences to a UMLProfile occurrence such as “UML analysis”.

Template is a subclass of Guidance. A Template is a predefined document that provides a standardized format for a particular kind of Artifact. The attribute *link* would normally be a URL referring to a local file.

Example of a Template:

“Microsoft Word template for Business Use Case Modeling”

Estimate is a subclass of Guidance. An Estimate describes an effort associated with a particular element. The description associated with an Estimate gives a context and interpretation for the effort.

9 Process Components

Figure **11** details the Process Components package. The classes in this package are concerned with dividing one or more process descriptions into self-contained parts that can be placed under configuration management and version control.

9.1 ProcessComponent

A ProcessComponent is a chunk of process description that is internally consistent and may be reused with other ProcessComponents to assemble a complete process.

A ProcessComponent contains a non-arbitrary set of ProcessDefinitionElements. Such a set must be self-contained; this means that there are no links from within the component to elements not within the component. It must be internally consistent in the sense that the multiplicities and constraints defined for the metamodel as a whole must be satisfied within the scope of the component.

Composition of ProcessComponents is done by a process of *unification*. For example, consider both of these:

- a ProcessComponent P1 that takes a set of high-level use cases and non-functional requirements as input and delivers an architecture as output
- a ProcessComponent P2 that takes an architecture and a set of detailed use cases as input, and delivers an executable, unit-tested body of code as output

To combine these two components, at least the output artifacts from P1 must be unified (that is, made identical) with the inputs to P2. Other elements may possibly be unified in addition, such as Templates, RoleKinds, and so on. Composition of ProcessComponents can only be fully automated if they originate from a common family so that the unification is obviously capable of being automated. If the components originate from different sources, the unification would involve human intervention that normally would consist of some re-writing of the elements, and possibly associated elements, to be unified–. Note that UPM permits both of these kinds of composition but provides no explicit support for either.

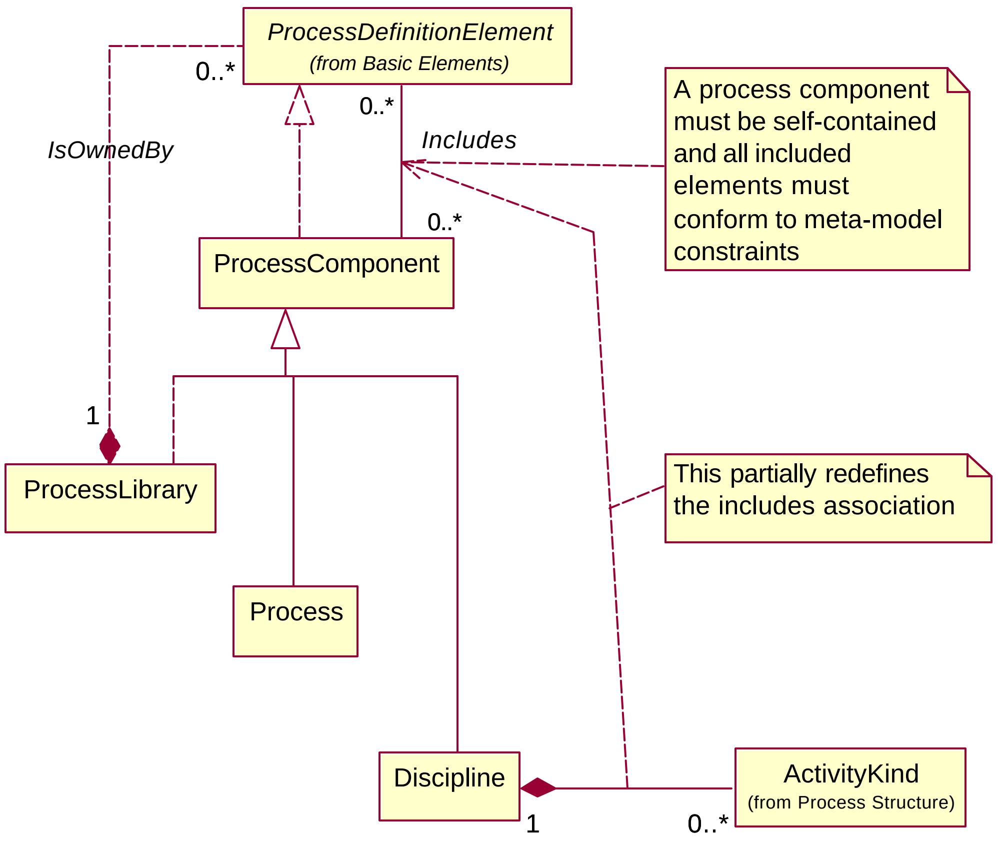


Figure 11—Process Components

9.2 *Process and ProcessFamily*

A Process is a ProcessComponent intended to stand alone as a complete, end-to-end process. It is distinguished from normal process components by the fact that it is not intended to be composed with other components. In a tooling context, the instance of Process is the "root" of the process model, from which a tool can start to compute the transitive closure of an entire process.

In addition to the "natural" relationships from the model, a process is defined through a Requires dependency that imposes the presence of a ProcessDefinitionElement when another ProcessDefinitionElement is part of the process.

A Lifecycle, as defined in Section 10.3, is also a specialization of a Process.

The class Process can also represent a family of processes, which is a process component out of which multiple overlapping processes can be defined.

9.3 Discipline

A *Discipline* is a particular specialization of ProcessComponent that partitions the ActivityKinds within a process according to a common "theme". Partitioning the ActivityKinds in this way implies that the associated RoleKinds, ArtifactKinds, and Guidance are similarly categorized under the theme. The composition between Discipline and ActivityKind partially redefines the *includes* association, in the sense that all ActivityKinds included in a Discipline are composed by the Discipline. In other words, the Disciplines partition the ActivityKinds and the ActivityKinds are lifetime dependent on the Disciplines.

Example

Nine disciplines are described in the Rational Unified Process 5.5: *Business Modeling, Requirement Management, Analysis & Design, Implementation, Test, Deployment, Project Management, Configuration and Change Management,* and *Environment*.

Synonyms

The IBM processes use the term 'domain'; the Rational Unified Process uses 'core workflow'; Objectory used 'process component'; Fusion uses the term 'phase'.

Notes

From the perspective of a static process description, the set of Disciplines used in a given process, ProcessFamily or ProcessComponent establishes a partitioning of all the static process definition elements. This means that no instance of any ProcessDefinitionElement should be left out outside of a discipline. High level introductory or reference material, such as introduction, glossary, definitions or bibliography may be included in an additional discipline.

9.4 ProcessLibrary

A ProcessLibrary provides another organization of ProcessComponents where a ProcessDefinitionElement belongs to one, and exactly one, ProcessLibrary. Process engineers use Process Libraries to manage the processes: define ownership, version, variants of processes or process families, as well as their delivery.

10 Process Lifecycle

In this package, shown in Figure 12, we introduce the process definition elements that define how the process will be run. They describe or constrain the behavior of the performing process, and are used to assist with planning, executing, and monitoring the process. As we stated earlier, a process can be seen as a collaboration between roles to achieve a certain goal or an objective. To guide its enactment, we need to indicate some order in which activities must be, or can be, executed. Also there is a need to define the “shape” of the process over time, and its lifecycle structure in terms of phases and iterations.

Note that these elements do not describe the enactment itself: they are elements in the process description that are used to help plan and execute enactments of that description.

For low-level, detailed, deterministic kinds of activities, it is tempting to document the workflow of a process as an activity diagram that shows strict sequences of activities or as sequence diagrams that show sequences of activities involving several roles. In this model, we are more concerned with identifying the set of activities required to achieve a certain goal, expressed in terms of states of artifacts, than precisely specifying the sequences of activities.

Nevertheless, if activity diagrams are required, UPM proposes to adopt the metamodel for them directly from the UML specification by creating a reference from the UPM class WorkItem to the UML metamodel class ActionState.

10.1 ArtifactStateSet

To each ArtifactKind we can associate a *ArtifactStateSet*. The ArtifactStateSet represents the lifecycle of the specific artifact as a set of ArtifactStates. Each artifact instance may have its own specific state set, often a trivial one with two states: ‘not done’ and ‘done’. A slightly more complex state set could have the states ‘created’, ‘under revision’, and ‘reviewed’. Transitions between states are not represented explicitly; if required, they can be represented using the Dependency class. ArtifactStateSets may be shared across multiple ArtifactKinds.

10.2 Condition, Goal, and Precondition

A *Condition* is defined as a set of artifact instances, each in a given state. This element is important to define the goals or objectives of a process or any WorkDefinition, such as major milestone, the objectives of an iteration, and so on. It is also used to define a precondition to an activity. A Condition has a set of ArtifactInStates each of which represents a particular artifact (by name) in a particular state. *Goal* and *Precondition* are the two subclasses of Condition, whereas condition is a specialization of ProcessDefinitionElement, which allows the expression of informal conditions as text.

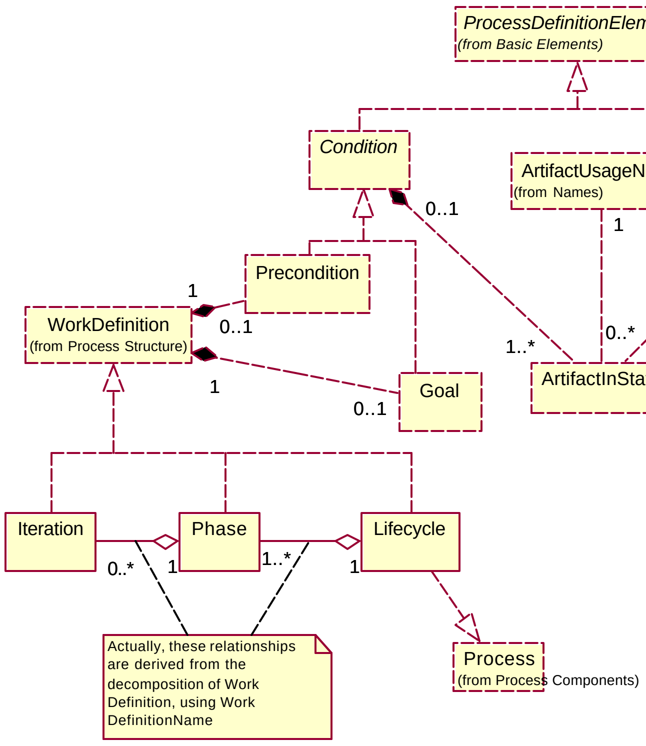


Figure 12—Process Lifecycle package

A WorkDefinition may have one Precondition that is defined when the work it describes can be executed. Each of a precondition's ArtifactInStates represents a particular input artifact (by name) in a particular state. This means that the WorkDefinition may not proceed unless all inputs are available in the correct state.

A WorkDefinition may have one Goal, which describes the state in which the artifacts it produces or modifies will be when it has finished execution. Each of a Goal's ArtifactInStates represents a particular output artifact (by name) in a particular state. This means that the WorkDefinition does not complete until all outputs are available in the correct state.

Synonyms

Processes also use terms such as ‘Entry criteria’, ‘Exit criteria’, ‘Success criteria’, and ‘Objectives’.

A Milestone is a goal for the completion of a major WorkDefinition, such as a Phase.

Constraints

- The ArtifactUsageName associated with an ArtifactInState must be contained within the containing WorkDefinition.
- An ArtifactInState is contained either by a Precondition or a Goal.

Examples

Barry Boehm [16] and the Rational Unified Process define four major milestones: *Lifecycle Objective (LCO), Lifecycle Architecture (LCA), Initial Operational Capability (IOC), and Product Release.*

Note

We do not use the formal term ‘post-condition’ because it is not normally used for a process, instead terms like ‘goal’, ‘objective’, and ‘milestone’ are used.

10.3 Phases, Iterations, and Lifecycle

A *Phase* is a specialization of WorkDefinition bounded by two conditions: a precondition that defines the entry criteria and a goal, called “Milestone” in this case, that defines the exit criteria. Phases are defined with the additional constraint of sequentiality; that is, they are executed with a series of milestone dates spread over time and often assume minimal (or no) overlap of their activities in time.

Examples

The Rational Unified Process (RUP) defines four sequential phases: *Inception, Elaboration, Construction*, and *Transition*. Moreover the RUP defines a phase as consisting of a certain number of iterations, which are workflows with minor milestones. OOSP has four phases: *Initiate, Construct, Deliver,* and *Maintain & Support* [15].

A process Lifecycle is defined as a sequence of Phases that achieve a specific goal. It defines the complete process to be enacted in a given project or program. It is a Process and is discussed further in Section 9.2.

11 Explanation of Enactment

This section provides further explanation to the UPM by introducing some classes that represent process enactment. This is done purely for the purpose of explanation. The classes in this section are not part of the formal UPM definition and are not part of the submission.

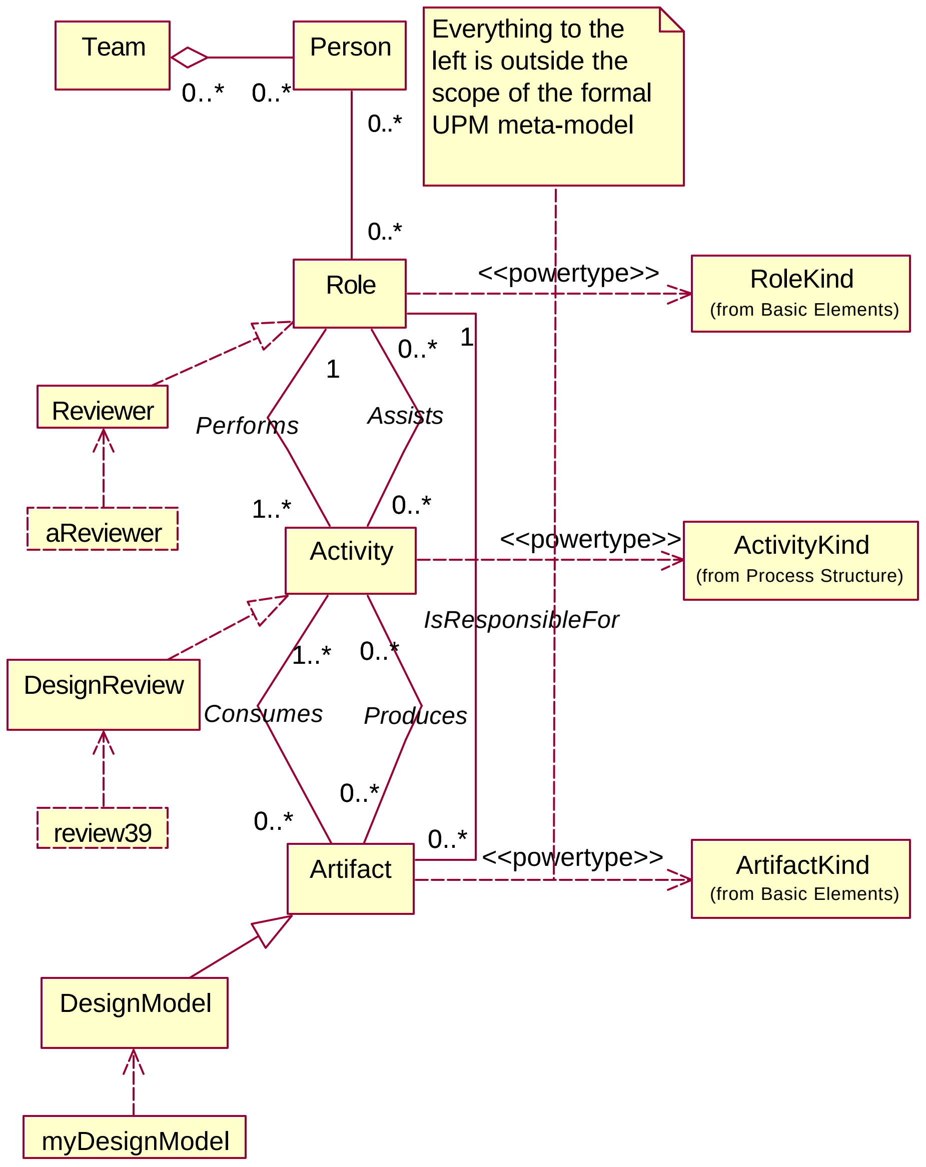


Figure 13—Enactment

The class Role represents the enactment of a RoleKind. For example, the RoleKind *Project Manager* might be enacted multiple times—in a given project the answer to the question “how many project managers are there on this project?” would be the number of instances of Role associated with the RoleKind *Project Manager*.

We have introduced a class Person to show that the association of Person to Role is many-many. A particular Person could play multiple roles; for example, Project Manager and Architect. A given Role could be carried out by a group of people; for example, Tester. The actual allocation of people to roles over time is a scheduling task and scheduling is outside the scope of this model.

It is useful to think of the association between Role and RoleKind as a *powertype* association. That is to say for each instance of RoleKind, there is a corresponding subclass of Role. In the example above, we would introduce a subclass of Role called ProjectManager. In Figure 13, we have shown a subclass of Role called Reviewer. These subclasses may possess their own attributes, associations and behavior.

In a similar way to Role, the class Activity represents the enactment of an ActivityKind. Figure 13 could be readily expanded to show the enactment of other subclasses of WorkDefinion. This area of the model could also be extended to show the resources associated with the enactment of particular activities such as tools, workstations, desks, chairs, and so forth. Modeling these resources and their allocation to actual activities on a project is a planning and scheduling task outside of the scope of this model.

Figure 13 shows a subclass of Activity called DesignReview and a subclass of Artifact called DesignModel. We also indicate specific instances of Reviewer, DesignReview, and DesignModel such as might be created (at level M0) in a particular project.

We could further extend this area of the model to show that a ProcessDefinitionElement is itself an Artifact. By doing so, we could define a process to describe how the process definition itself is created or modified.

12 Management of Process Assets

The management of multiple processes, variants, derivatives or versions is beyond the scope of this metamodel. As all techniques and tools used in the area of configuration management and change management for software can be applied literally to a software process product, it does not make sense to replicate these aspects in the UPM. See standards IEEE 610.12-1990 or ISO 12207.

All ProcessDefinitionElements are *configuration items*. As such, they can have multiple *versions*. The versions of a given configuration item are linked to each other to form *histories*. *Variants* can be introduced by creating parallel histories. A specific *process configuration* is formed by selecting one version, at the most, for each ProcessDefinitionElement. If a process definition element is required in two forms within a single process configuration, it must be cloned and given a specific identity; for example, “simple design review” versus a “complex and critical review”. Process variants are defined similarly by selecting ProcessDefinitionElements from a consistent set of version histories all belonging to the same variant. ProcessLibraries can be used for this purpose.

13 UPM, UML, and MOF

In this section, we discuss the following two topics:

1. the fact that UPM is not defined as a UML profile
2. the positioning of UPM in the four-layer architecture

13.1 UPM as a UML Profile

The RFP asks for the metamodel to be defined, if possible, as a UML profile. This submission does not do so.

According to the paper Requirements for UML Profiles (OMG document ad/99-12-32): "From a comparative perspective, UML Profiles form a metamodel extension mechanism that imposes certain restrictions on how the UML metamodel can be modified. For instance, it is not possible to insert new metaclasses in the UML metaclass hierarchy (i.e. new superclasses for standard UML metaclasses). It is also not possible to modify the standard UML metaclass definitions, e.g. by adding meta-associations. Such restrictions do not apply in MOF context: there in principle any metamodel can be registered with a repository, and they can completely bypass or even ‘botch up’ UML (they can, however, also apply the same restrictions as profiles embody)."

Therefore, for UPM to be a UML profile, it would be necessary for each UPM class to identify a class in the UML 1.3 metamodel, either for use directly or from which the UPM class could be derived by stereotyping. Furthermore, it would be necessary to model every UPM association by an existing UML metamodel association.

As a starting hypothesis, we would consider unifying the UPM ProcessDefinitionElement with UML ModelElement. Given this hypothesis, here we can identify some of the cases where we have not found a reasonable element in the UML definition to support UPM requirements.
Naming
In several places in UPM it is possible for a process definition element to have different names in multiple contexts. Unfortunately, the UML namespace itself does not offer a way for a model element to have different names in different contexts. A possible pattern for this in UML is Parameter but to use a Parameter requires that the context in which the name occurs is a kind of BehaviouralFeature and the element named is a Classifier. We see no straightforward way of mapping UPM constructs to match this pattern.
Support for Natural Languages
UML provides no support for multiple natural languages such as internal and external names.

Operations

To correspond to the conceptual model of UPM illustrated in Figure **2**, we could potentially unify WorkItem with UML Operation and RoleKind with Class. However, doing this offers no obvious way to represent the decomposition of WorkDefinition using WorkDefinitionName. This could, perhaps, be done by means of special parameters but the resulting model would be obscure to say the least.

This short list of issues is by no means exhaustive but serves to illustrate that rendering UPM as a UML profile, although conceivably possible by applying numerous workarounds, would not give rise to an elegant or readily understandable proposal. We propose to continue to work on this issue between now and the time of the final submission, because there are significant advantages in formulating the UPM as a UML profile, mainly due to:

(a) the ability to use existing UML tools for process modeling
(b) the ability to leverage UML notation directly.

A different approach for extending UML has been adopted within the OMG in the Common Warehouse Model (CWM) specification (OMG document ad / 2000-01-01). This specification defines an extension to UML by using MOF facilities directly, rather than UML profile facilities. This is done according to the principle that every class within the CWM definition inherits directly or indirectly from some class in the UML definition. It is claimed in the CWM definition that this permits the use of UML notation in the diagrammatic representations of the CWM metamodel. It is straightforward in most cases to see how this would work, although no normative definition exists of a mapping from UML notational elements to CWM classes.

Therefore, an alternative for UPM is to adopt the CWM philosophy for extending UML. This is a fallback position that we will adopt in the final submission should it not prove feasible to map UPM as a UML profile.

Ultimately, these dilemmas point to more basic problems. First, we would ideally like to be able to reuse a set of generic metamodelling patterns such as the general ability for a container to contain elements and for elements to be referred to by different names in different contexts. The current UML definition does not provide such patterns, however, the work being done toward issuing an RFP for UML 2 indicates that a future version of UML might. Second, we would like a more flexible and precisely defined way to map from metamodel elements defined in MOF to their notational counterparts. Once again, this is an issue under consideration in the preparation of the RFP for UML 2.

13.2 Positioning in Four-layer Model

The UPM metamodel clearly fits at level M2 in the four-layer model and there is no problem with this for the purpose of the current submission. However, future extensions to this model intended for modeling process enactment do give rise to some significant issues in positioning. For this discussion we refer back to the example in Figure **13**.

In this example the class RoleKind represents descriptions of Roles. For example, there will be an instance of a RoleKind describing Reviewer and saying things about what a Reviewer needs to do. RoleKind is a powertype of Role, which means that for every instance of RoleKind there is a subclass of Role. Therefore, a subclass of Role called Reviewer exists and there will be an instance of Reviewer for every situation where this role is actually played (or is planned to be played).

The issue is how can this example fit into the four-layer architecture? There seems to be a number of possible choices, none of which are entirely satisfactory and none of which are properly supported by the current architecture.

1. RoleKind is at M2 and Role is at M1. This corresponds to the RFP, which states that models of specific processes exist at M1 while their metamodels

exist at M2.

Unfortunately there is no mechanism in the OMG architecture to represent the concept of powertype between layers. Therefore, we must forget about making RoleKind a powertype of Role. We can still create classes at M1 called Reviewer, ProjectManager, and so on, but they will not inherit from a common superclass. We do not seem to have any way to define Role. Given that most Roles will want to share the same behavior with regard to planning, scheduling, associating with activities, and so on this lack of a common superclass is a serious problem. This could be resolved while retaining the layered positioning by extending MOF so that M2 elements can act as powertypes for M1 elements.

2. Both RoleKind and Role are at M1. This resolves the problem of the definition of powertype because UML (at M2) defines what a powertype is.

   This conflicts with the architectural positioning mandated in the RFP, which requires the UPM metamodel to be at M2. Also, it leads to a problem because, elsewhere in the model, we have a class called "UMLProfile" that reifies the notion of a UML profile. How can a model in UML (at M1) refer to a reified UML Profile that conceptually seems to belong at M3?

3. RoleKind, Role, and Reviewer are all at M2, whereas MOF is extended to incorporate the notion of a powertype between M2 elements.

   In some ways this seems to make life easier. Then again we have to ask what happens at M1? Instances of RoleKind ought to appear at M1, however, they are actually appearing at M2.

In summary, although the current UPM submission can be readily positioned at M2, the positioning of probable extensions to UPM in the four-layer model is problematical and seems to require modifications to the layered architecture itself. We note that such modifications are within the scope of the UML 2 Request for Proposal currently being drafted and we, therefore, propose that the requirements for supporting the powertype pattern discussed here should be included in that RFP.

14 Glossary

Activity

In the enacted process, an activity is a piece of work executed by one role. The granularity of an activity is usually a few person-days.

Activity Kind

A Work Definition describing what takes place in one kind of activity. Activities are the main element of work. See also Activity Group and Steps.

Artifact

In the enacted process, an artifact is any piece of information or physical entity produced or used by the activities of the software engineering process. Examples of artifacts include models, plans, code, executables, documents, databases, and so on.

Artifact Kind

A Process Definition Element describing one kind of artifact. Note that ArtifactKind is the powertype that ranges over the class Artifact.

Component (*see* Process Component)

Dependency

A Dependency is a process-specific relationship between Process Definition Elements. Examples of dependencies include *Tracing Dependency* between Artifact Kinds and *Precedence Dependency* between Work Definitions such as Activity Kinds. The *Requires* dependency forces a ProcessDefinitionElement to be part of the same Process or ProcessFamily as another.

Discipline

A discipline is a process component organized from the perspective of one of the software engineering disciplines: Configuration Management, Analysis & Design, and so forth.

Element (*see* Process Definition Element)

Goal

A goal is a specific condition that is satisfied at the end of a given Work Definition such as the end of an activity group or a phase.

Guidance

Guidance is a Process Definition Element associated to the major process definition elements, which contains additional descriptions such as techniques, guidelines and UML profiles, procedures, standards, templates of artifacts, examples of artifacts, definitions, and so on.

Iteration

An Iteration is a large-grained Work Definition that represents a complete pass through the software engineering process and results in a release (internal or external) of the software product.

Milestone

A milestone is a synonym for the goal of a phase**.**

Name

A Name is a way to explicitly denote a Process Definition Element. Names are used in some process definition elements to designate other process definition elements. Example of names are Input and Output of Work Definition, or and the names of contained artifacts within their containing artifacts.

Phase

A high-level Work Definition, bounded by a Milestone.

Process Component

A Process Component is a coherent aggregate of Process Definition Elements organized from a given vantage point such as a discipline, for example, testing, or the production of some specific artifact, for example, requirements management.

Process Definition Element

An element describing one aspect of a software engineering process.

Process

A Process is a Process Component that contains a complete description of a software engineering process.

Process Family

A Process Family is a set of Process Definition Elements that satisfy the Requires dependency.

Process Library

A Process Library is a set of ProcessDefinitionElements that have a common ownership. Every element is owned by only one Process Library.

Role Kind

A Role Kind is a process definition element describing the roles, responsibilities, and competence of one kind of role.

Role

A role denotes one of several roles that may be played by an individual (or a small group of individuals) in the process. In the enacted process, roles “execute” the activities.

State

An attribute of an artifact, defined in the State set associated to the corresponding Artifact Kind.

State Set

Associated to an artifact kind, a state set defines the states that the corresponding artifact can take during the process.

Steps

An atomic and fine-grained Work Definition Element used to decompose Activities. Activities are partially ordered sets of steps.

Work Definition

A process definition element describing the execution, the operations performed, and the transformations operated on the artifacts by the roles. Steps, Activity, Activity Group, Iteration, Phase, and Lifecycle are kinds of work definition.

15 References

This section is not by any means intended to cover the whole literature on process and process modeling (see the extensive bibliography given in [6]), but to give the principal sources we have used in elaborating this specification.

1. *Rational Unified Process* (RUP) 2000, Rational Software Corporation, Cupertino, CA (2000)
2. Ivar Jacobson, et al., *Object-oriented Software Engineering—A Use Case Driven Approach*, Addison-Wesley (1992).
3. Philippe Kruchten, *The Rational Unified Process—An Introduction*, 2nd ed, Addison-Wesley-Longman, Reading, MA (2000)
4. Ian Graham, Brian Henderson-Sellers, and HoumanYounessi, *The OPEN Process Specification*, Addison-Wesley, London, UK, 1997, 314pp
5. B. Henderson-Sellers, S. Mellor, "Tailoring process methodologies," *ROAD/JOOP*, Volume 12 no 4, July / Aug 1999 (?)
6. Jean-Claude Derniame, et al., *Software Process: Principles, Methodology, and Technology*, LNCS #1500, Springer-Verlag, 1999.
7. Ivar Jacobson, Grady Booch, Jim Rumbaugh, *The Unified Software Development Process*, Addison-Wesley-Longman (1999)
8. Grady Booch, et al., *UML User's Guide*, Addison-Wesley-Longman, Reading, MA (1999)
9. Desmond D'Souza & Alan Wills, *Objects, Components and Framework with UML—The Catalysis Approach*, Addison-Wesley–Longman (1998)
10. Jennifer Stapleton, *DSDM—Dynamic Systems Development Method*, Addison-Wesley (1998)
11. Walker Royce, *Software Project Management—A Unified Framework*, Addison-Wesley-Longman (1998)
12. IBM, *Developing Object-Oriented Software—An Experience-Based Approach*, Prentice-Hall (1997)
13. Derek Coleman et al., *OOD—The Fusion Method*, Prentice-Hall (1994)
14. Alfonso Fuggetta & Alexander Wolf (eds.), *Software Process*, J. Wiley & Sons (1996)
15. Scott Ambler, *Process Patterns—Building Large-Scale Systems using Object technology*, SIGS Books Cambridge University Press (1998)
16. Barry Boehm, "Anchoring the Software Process," *IEEE Software*, July 1996, 73-82.
17. OMG, *RFP for Software Process Engineering*, 1.0, November 1999
18. 12207 ISO / IEC 12207 Information technology – Software life-cycle processes, ISO, Geneva, 1995
19. IEEE 1074-1997, *Standard for developing software life cycle processes*, NY, NY 1997
20. I. Jacobson and S. Jacobson, " Reengineering your software engineering process" *Object Magazine*, March 1995.
21. C. Larman, *Applying UML and Patterns—An Introduction to Object-Oriented Analysis and Design*, Prentice-Hall (1997)
22. S. Cook and J. Daniels, *Designing Object Systems: object-oriented Modeling with Syntropy*, Prentice-Hall (1994)

Annex 1: Translation table

This annex maps the terminology from different sources:

UPM	*Rational Unified Process 5.5 [1]*	*IBM SI & WSDDM [12]*	*OPEN [4]*	*OOSP [15]*	*Promoter [6]*	*IEEE 1074-1997 [19]*	*ISO/IEC 12207 [18]*
Lifecycle	Process	Engagement model Process shape	Lifecycle process		Lifecycle	Lifecycle process	Lifecycle model
Phase	Phase	Phase	Stages	Phase		Phase	Phase
WorkflowKind	Iteration	Increment Activity		Iteration Stage WBS			Activity
Milestone	Milestone						
Discipline	Core workflow	Domain (SI) Phase (Wisdom)	Activity			Activity group	Process
RoleKind	Worker	Role	Rôle		Role		Role
ActivityKind	Activity Step	Task	Task Subtask	Task Activity	Activity	Activity	Task
ArtifactKind	Artifact	Workproduct	Deliverable	Deliverable	Product	Product	Artifact Software product Lifecycle data
GuidanceKind	Guidelines Tool mentors Templates….	Technique Procedure	Technique	Guideline Standard	Direction		
Person	Person	Person	Resource		Performer Agent		Resource
Team	Team				Organization	Organization	Developer
ToolKind	Tool				Tool		

Annex 2: Overview of the model

This class diagram summarizes the main classes in the Unified Process Model.

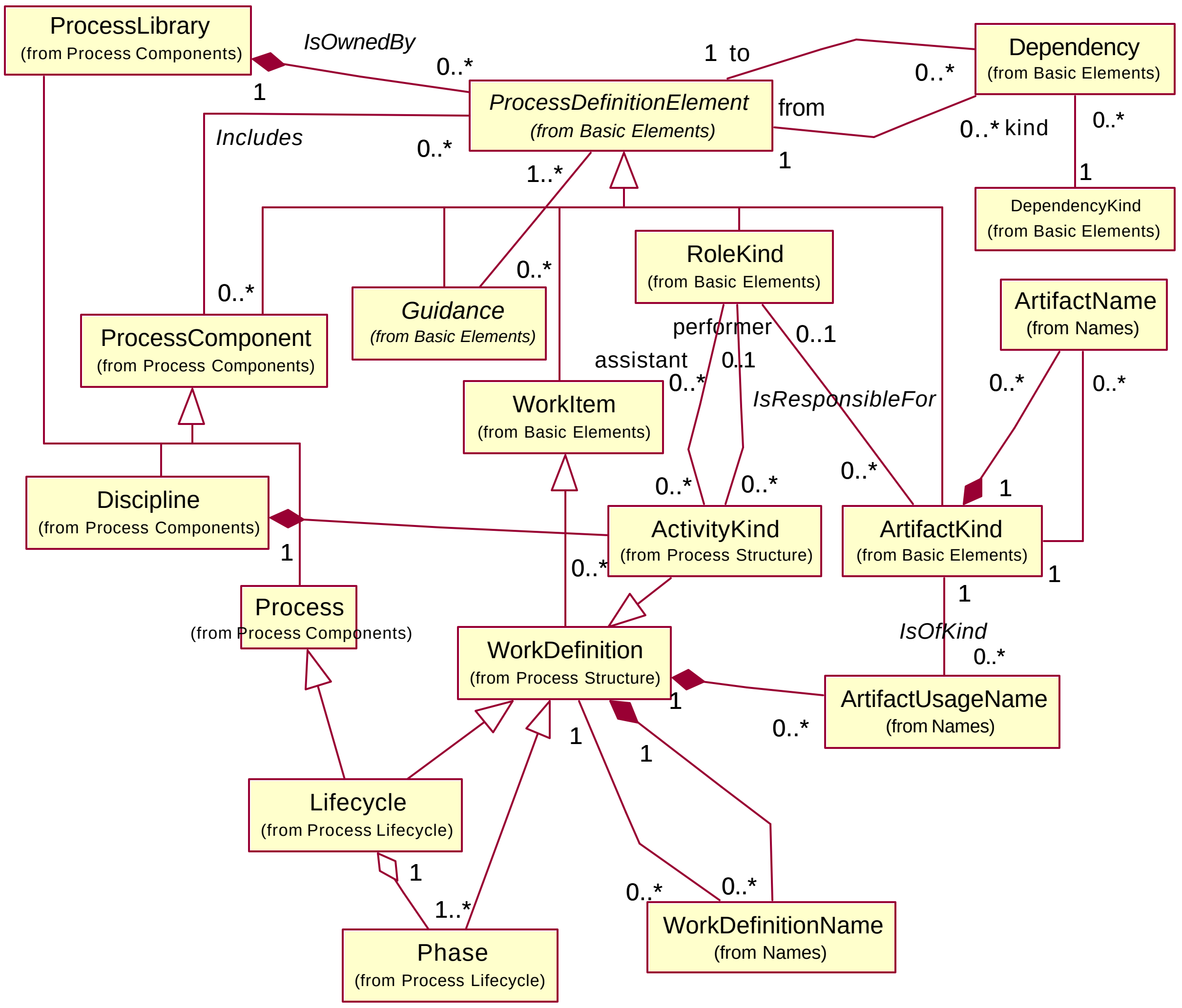
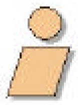
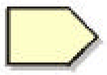
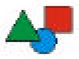
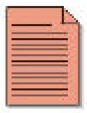
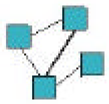
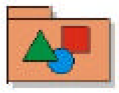
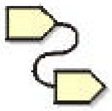
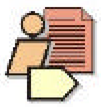
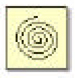


Figure 14—Overview of the UPM model

Annex 3: Proposed Graphical Notation

This section suggests how a process notation for UPM could look like, using UML-like diagrams.

UPM Element	*Icon*	*Comments*
RoleKind		
ActivityKind		
ArtifactKind	   	Simple artifact, Document Model Aggregate of artifacts
WorkDefinition		Other than ActivityKind
ProcessComponent		
Process		

Diagrams similar to UML class diagrams could be used to represent ArtifactKind composition or dependencies, work breakdown structure or groupings in ProcessComponents, as shown in sample Figure 15. In this suggested diagram, the name and the entity designated by this name have been collapsed in a single pictorial element to make the diagram less cluttered and more legible.

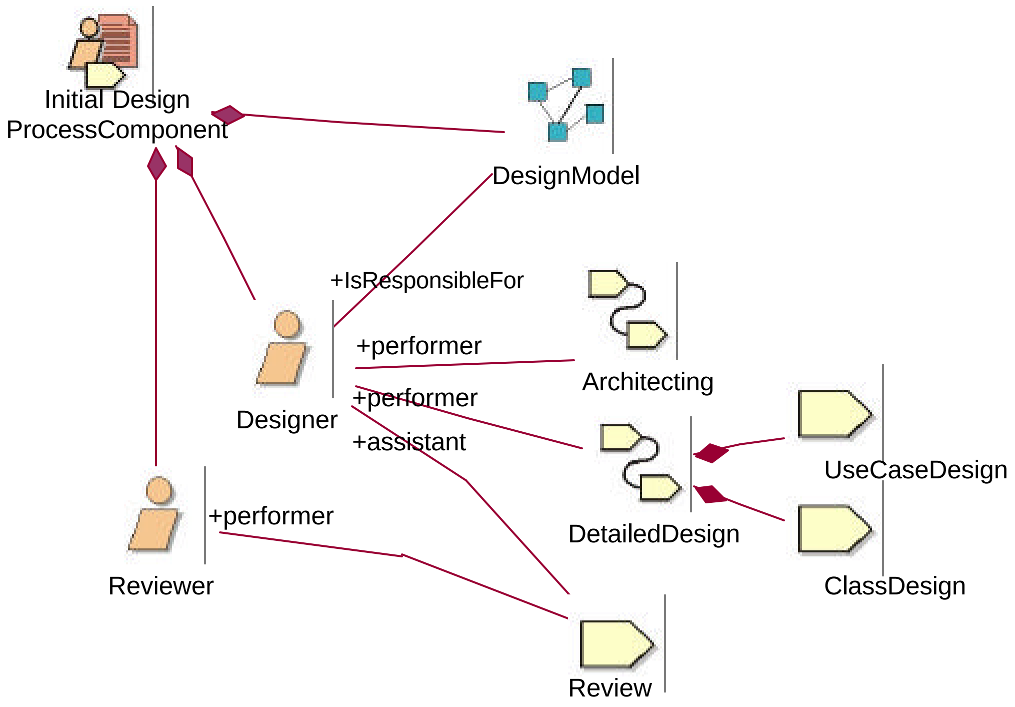


Figure 15—Example of process component

Diagrams like UML activity diagrams could be used to represent sequencing of WorkItems (see sample Figure 16).

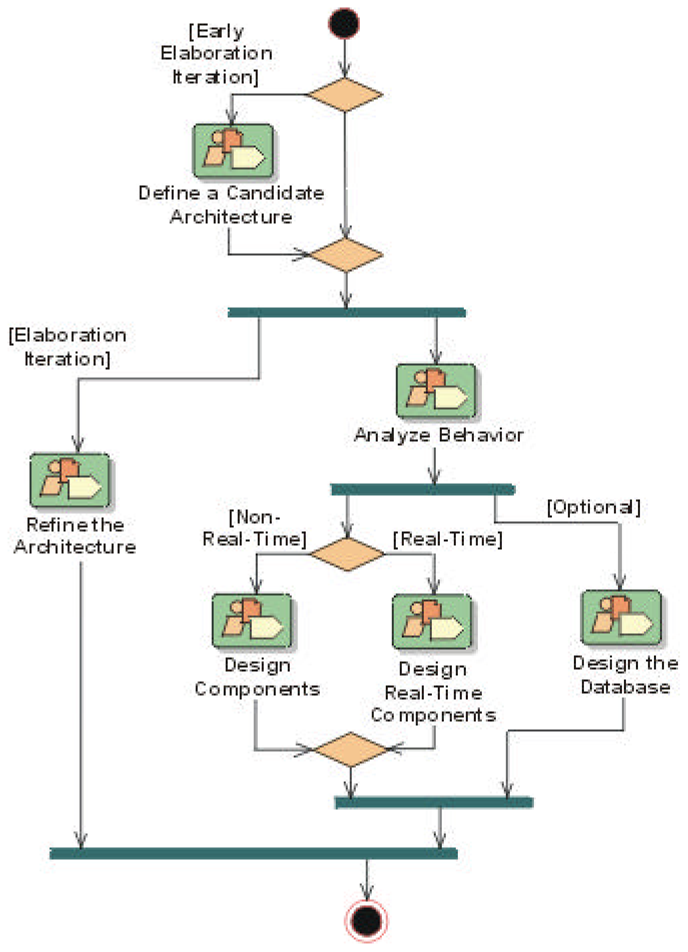


Figure 16—Activity diagram