Software Process Engineering Management

The Software Process Engineering Metamodel (SPEM)

Second Revised Submission

OMG document number: ad/2001-06-05

June 6th, 2001

Submitted by
IBM
Rational Software
Fujitsu/DMR
SOFTEAM
Unisys
Alcatel
Q-Labs

Supported by
Valtech
Toshiba
Siemens
Computer Associates
Adaptive Ltd.
Nihon Unisys Ltd.
# Table of Contents

1 **Introduction** ................................................................................................................. 6
   1.1 **Overview** .................................................................................................................. 6
   1.2 **Modeling Approach** .................................................................................................. 6
   1.3 **Scope** ......................................................................................................................... 7
   1.4 **Terminology** .............................................................................................................. 7
   1.5 **Relationships to Other OMG Specifications** ............................................................. 8
       UML ................................................................................................................................. 8
       UML Profile .................................................................................................................... 8
       MOF 1.3 and XMI ............................................................................................................ 9
       Workflow ........................................................................................................................ 10
       Proof of Concept ........................................................................................................... 10
   1.6 **Compliance points** ................................................................................................... 11
       Examples ......................................................................................................................... 12

2 **Mapping to RFP Requirements** ..................................................................................... 13
   2.1 **Mandatory Requirements:** ......................................................................................... 13
       Four-layer Architecture .................................................................................................. 13
       Relationship to UML and MOF ..................................................................................... 13
       XMI DTD ........................................................................................................................ 13
       Basic Concepts .............................................................................................................. 13
       Process Examples ......................................................................................................... 14
       Process Patterns and/or Components .......................................................................... 14
       Glossary .......................................................................................................................... 14
       Support for UML .......................................................................................................... 14
       Categories ...................................................................................................................... 14
       Natural Language Translation ..................................................................................... 15
       Graphical Notation ....................................................................................................... 15
   2.2 **Optional Requirements** ................................................................................................ 15
       Submission as a UML Profile .......................................................................................... 15
       Definition of Process Patterns and/or Components ...................................................... 16
       Reification of UML Profile Concept ............................................................................. 16

3 **SPEM Foundation** ........................................................................................................... 17
   3.1 **SPEM/Foundation::Data_Types** .................................................................................. 17
   3.2 **SPEM/Foundation::Core** ............................................................................................ 18
   3.3 **SPEM/Foundation::Model_Management** .................................................................. 21
   3.4 **SPEM/Foundation Well-Formedness Rules** .............................................................. 22

4 **Conceptual Model** .......................................................................................................... 23

5 **Package Structure** ......................................................................................................... 24

6 **Basic Elements** ............................................................................................................... 25
   6.1 **ExternalDescription** .................................................................................................. 25
6.2 Guidance ............................................................................................................................. 25
   Kinds of Guidance .............................................................................................................. 26

7 Dependencies .................................................................................................................... 28
   7.1 SPEM Dependencies .................................................................................................... 28
   7.2 Well-formedness rules ................................................................................................. 30

8 Process Structure .............................................................................................................. 31
   8.1 WorkProduct and InformationElement .................................................................. 31
      Associations .................................................................................................................. 31
      Attributes .................................................................................................................... 32
      Note ............................................................................................................................. 32
      Examples ...................................................................................................................... 32
      Synonyms .................................................................................................................... 32
   8.2 WorkDefinition and ActivityParameter .................................................................. 32
      Associations .................................................................................................................. 32
      Attributes .................................................................................................................... 33
      Note ............................................................................................................................. 33
      Example ....................................................................................................................... 33
   8.3 Activity and Step ........................................................................................................ 33
      Associations .................................................................................................................. 33
      Examples ...................................................................................................................... 34
      Synonyms .................................................................................................................... 34
   8.4 ProcessPerformer and ProcessRole ....................................................................... 34
      Associations .................................................................................................................. 34
      Synonyms .................................................................................................................... 34
      Examples ...................................................................................................................... 35
      Notes ........................................................................................................................... 35
   8.5 Well-formedness rules ............................................................................................... 35

9 Process Components ........................................................................................................ 37
   9.1 Package ....................................................................................................................... 37
   9.2 ProcessComponent ..................................................................................................... 37
      Example ....................................................................................................................... 37
   9.3 Process ......................................................................................................................... 38
   9.4 Discipline .................................................................................................................... 39
      Example ....................................................................................................................... 39
      Synonyms .................................................................................................................... 39
   9.5 Well-formedness rules ............................................................................................... 39

10 Process Lifecycle ............................................................................................................ 41
   10.1 Phase ......................................................................................................................... 41
      Examples ...................................................................................................................... 42
   10.2 Lifecycle ..................................................................................................................... 42
      Associations .................................................................................................................. 42
      Example ....................................................................................................................... 42
   10.3 Iteration ..................................................................................................................... 42
      Example ....................................................................................................................... 42
1 Introduction

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

- IBM Corporation, Steve Cook  
scook@acm.org
- Rational Software, Philippe Kruchten  
pbk@rational.com
- Fujitsu/DMR, Pierre Montminy and Hiroshi Miyazaki  
Pierre_Montminy@dmr.ca  
miyazaki.hir-02@jp.fujitsu.com
- SOFTEAM, Philippe Desfray  
philippe.desfray@softeam.fr
- Unisys, Sridhar Iyengar  
Sridhar.Iyengar2@unisys.com
- Alcatel, Laurent Rioux  
Laurent.Rioux@alcatel.fr
- Q-Labs, Annie Kunzmann-Combelles  
akc@q-labs.fr

We also acknowledge support from:

- Valtech, Craig Larman  
craig.larman@valtech.com
- Toshiba, Mari Natori  
marin@sitc.toshiba.co.jp
- Siemens, Olaf Kaestner  
olaf@sni-svy.com
- Computer Associates, Alan Birchenough  
alan.birchenough@ca.com
- Adaptive Ltd, Pete Rivett  
pete.rivett@adaptive.com
- Nihon Unisys Ltd, Kiyoshi Sakaguchi  
Kiyoshi.Sakaguchi@unisys.co.jp

Finally we thank for their contributions Alan Bradbury (Adaptive), Donald Baisley (Unisys), who produced the DTDs, as well as Van-Si Nguyen (Xerox), Steve Tockey (Construx), Gail Trotter (Boeing), Pierre N. Robillard (Université de Montréal), Brian Henderson-Sellers (University of Technology, Sydney), Mariano Belaunde (France Telecom), Björn Gustafsson (Rational Software), John Cameron, Ed Kahan, Dan D’Elena (IBM), Ed Ferrara, Phillip Rossomando (Unisys), and Gilbert Raymond (Softeam) for their constructive criticisms.

1.1 Overview

This document presents the Software Process Engineering Metamodel (SPEM). This metamodel is used to describe a concrete software development process or a family of related software development processes. Process enactment is outside the scope of SPEM, 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 1-1 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 2001 (RUP2001), DMR Macroscope or the IBM Global Services Method
are 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.

The SPEM specification is structured as a UML profile, and also provides a
complete MOF-based metamodel. This approach facilitates exchange with both
UML tools and MOF-based tools/repositories.

1.3 Scope
The SPEM is a metamodel for defining processes and their components. A tool
based on SPEM 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 SPEM, is not in the scope of this model.
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 Global Services 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 Chapter 15 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. UML 1.4 (January 2001) is the version referred to throughout this document.

The UML is defined by a metamodel, which is itself defined as an instance of the MOF (Meta-Object Facility) metamodel. A subset of the UML graphical notation is used to depict this metamodel. The SPEM metamodel is defined similarly, and is formally defined as an extension of a subset of UML called SPEM_Foundation. Chapter 3 describes SPEM_Foundation in detail.

The purpose of the Software Process Engineering Model (SPEM) 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 kind of variant of UML that uses the extension mechanisms of UML in a standardized way, for a particular purpose.

The UML 1.4 semantics (OMG document ad/01-02-13)) provides the following definition in the section “2.14.4 Semantics”:

A profile stereotype of Package contains one or more related extensions of standard UML semantics (refer to Section 2.6, “Extension Mechanisms”). These are normally intended to customize UML for a particular domain or purpose. Profiles can contain stereotypes, tag definitions, and constraints. They
can also contain data types that are used by tag definitions for informally declaring the types of the values that can be associated with tag definitions.

In addition, a profile package can specify a related model library and identify a subset of the UML metamodel that is applicable for the profile. In principle, profiles merely refine the standard semantics of UML by adding further constraints and interpretations that capture domain-specific semantics and modeling patterns. They do not add any new fundamental concepts.

The SPEM is defined both as a metamodel and as a UML profile, which allows SPEM modelers to use the UML as a concrete notation. Chapter 12 of this proposal discusses the profile.

**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.

In addition to defining SPEM as a UML profile, it is defined as a MOF metamodel, based on a subset of UML. This gives a more restricted version of SPEM, in which the basic SPE elements can be described, without some of the diagramming and structuring facilities, which are added by the profile version of SPEM. Chapter 12 describes the additional facilities gained when SPEM is treated as a UML profile.

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.

XMI can be used to manipulate the SPEM metamodel as follows:
• to create a SPEM Document Type Definition
• to transfer process models based on SPEM as XML documents, either by describing the model as a direct SPEM instance (usage of the SPEM DTD) or by describing it as a UML model conforming to the UML profile for SPEM (usage of the UML DTD)
• to transform the SPEM metamodel itself into an XML document, based on the MOF DTD, for interchange between MOF-compliant repositories.

Chapter 14 of this document describes the use of XMI to interchange SPEM-based models.

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 somewhat overlaps the scope of the current submission.

Proof of Concept
The (meta)model and the UML Profile presented here supports at least the Rational Unified Process, DMR Macroscopic, IBM's Global Services Method and the Unisys QuadCycle method. Examples throughout the text show how particular elements in the model are used in these and other processes. The
SPEM is supported by the Rational Process Workbench (RPW), which is a process authoring tool based on UML. The SPEM profile has been implemented using the “Objecteering/UML Profile Builder” tool of SOFTEAM, and then applied to the “Objecteering/UML Modeler” tool, which has been used as a “SPEM modeler” to represent various processes. All the SPEM extensions have been implemented with most of the SPEM well-formedness rules. The SPEM metamodel server has been generated in the Unisys XMI/MOF tools. Finally see Annex 2 for an example based on the DMR Macroscope.

1.6 Compliance points

When specifying their compliance to SPEM, vendors should refer to the compliance points defined in this section, and not loosely say they are “SPEM compliant.” Being compliant to one point means that all elements belonging to this point are implemented. As a general rule, all elements defined in the SPEM metamodel (chapters 5 to 10) shall be supported except for the following optional elements:

- Kinds of Guidance (see section 6.2)
- Steps (see section 8.3)
- InformationElement (see section 8.1)
- Discipline (see section 9.4)

Also it is not mandated that a SPEM implementation use the same terminology. Other terminologies, and natural languages other than English, can be used. In this case, a correspondence list must present a mapping of this terminology with the SPEM terminology.

The compliance points are as follow:

- **UML Profile for SPEM**: the compliant implementation shall implement all the UML parts extended by SPEM, and shall define all the SPEM extensions. The compliant specification should specify whether it implements the SPEM constraints by an automated check or not. A SPEM Profile compliant implementation shall provide the UML XMI exchange mechanism that supports all UML features extended by SPEM, and the UML extension mechanism (UML Profiles).
- **Metamodel**: the compliant implementation shall support the SPEM Metamodel, except possibly some of the optional elements as noted above.
- **MOF/XMI DTD**: The compliant specification should implement all the MOF based metamodel provided by the SPEM specification. It shall implement the XMI DTD specified by the SPEM standard.
- **Notation**: The compliant implementation shall recognizably support all the notation defined by the SPEM specification.

Any combination of the four compliance points can be used.
Examples

Implementers declare their SPEM compliance in the following form:

- The XXX tool is SPEM compliant (UML Profile for SPEM without constraint checks implementation, Notation).
- The XXX tool is SPEM compliant (Metamodel, MOF/XMI DTD, Notation).
- The XXX tool is SPEM compliant (Notation).

This list is not exhaustive.
2 Mapping to RFP Requirements

2.1 Mandatory Requirements:

Four-layer Architecture

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

SPEM sits at level M2 in the four-layer architecture and further details are found in section 1.2 and chapter 12.

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 SPEM metamodel is defined using a subset of UML notation in a similar way to UML itself (more precisely, to the physical model defined in the UML 1.4 specification) and to MOF. This subset of UML notation corresponds to the facilities supported by MOF and its semantics are defined by the MOF 1.3 specification.

XMI DTD

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

An XMI DTD is included in chapter 14.

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 SPEM supports the description of these concepts (not their enactment) in the following ways:

- Tasks are modeled by Activity (see section 8.3)
- Techniques are modeled by Guidance (see section 6.2) or Activity (see section 8.3)
- Roles are modeled by ProcessRole (see section 8.4)
- Products are modeled by WorkProduct (see section 8.1)
- Phases are modeled by Phase (see section 10.1)
Annex 1 shows a mapping of SPEM terms with those of various published processes.

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

SPEM underpins the Rational Unified Process and IBM's family of methods, including the Global Services Method deployed throughout IBM Global Services. See also an example extracted from the DMR Macroscope in Annex 2.

**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. Also the construct ProcessPerformer, described in more detail in section 8.4, represents a reusable set of WorkDefinitions and Activities.

**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.**

Chapter 15 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 SPEM 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 class Package, explained in section 9.1, supports a categorization of process elements. By using the categorizes dependency described in chapter 7, it supports multiple, overlapping categorization schemes. The class Discipline supports a categorization based on a partitioning of the Activities.

**Natural Language Translation**

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

With each ModelElement can be associated one or more ExternalDescription. Each ExternalDescription has an attribute ‘language’ specifying which natural language is used for the name and the description of the ModelElement. See section 6.1.

**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 SPEM offers graphical notations that are similar to those of UML for depicting software engineering processes.

Special icons are used to denote process-related concepts—artifacts, roles, activities, and so on—to make these diagrams more expressive. Chapter 13 of this document defines a notation for SPEM process modeling that extends UML notation with process-related icons.

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.

As well as being defined by a stand-alone metamodel, this submission is presented as a UML profile. Chapter 12 discusses this profile and shows how the
constructs in the stand-alone metamodel are mapped into the profile constructs. Note that the UML profile provides significantly more facilities than the stand-alone metamodel, in particular the use of various notations.

**Definition of Process Patterns and/or Components**

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

Examples of actual process components are included in the SPEM, in Annex 2.

**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.

A UML Profile can be represented as a kind of Guidance, as described in section 6.2.
3 SPEM Foundation

The SPEM stand-alone metamodel is built by extending a subset of the UML 1.4 physical metamodel. This UML subset is called SPEM_Foundation, as shown in Figure 3-1. This chapter describes the content of the SPEM_Foundation package.

![Figure 3-1 - the SPEM_Foundation and SPEM_Extensions packages](image)

3.1 SPEM_Foundation::Data_Types

The SPEM_Foundation::Data_Types package is a subset of the UML 1.4 Data_Types package, and contains definitions of the following data types as shown in Figure 3-2: Integer, UnlimitedInteger, String, AggregationKind, Boolean, ParameterDirectionKind, Name, Multiplicity and MultiplicityRange. The Data_Types package also contains definitions of Expression and BooleanExpression as shown in Figure 3-3. The SPEM Foundation data types and expressions are defined exactly as in UML 1.4 section 2.4.
Figure 3-2 - Foundation Data Types Package - Data Types

Figure 3-3 - Foundation Data Types Package - Expressions

3.2 SPEM_Foundation::Core

The SPEM_Foundation::Core package is structured similarly to the UML 1.4 Core packages and is shown diagrammatically in the following figures. Figure 3-4 shows the model elements that form the structural backbone of the metamodel. Figure 3-5 shows the model elements that define relationships. Figure 3-6 shows the model elements that define dependencies. Figure 3-7 shows the model elements that define auxiliary elements.

In each case, classes have been omitted from the UML 1.4 metamodel, and in many cases, attributes have been omitted from included classes. What remains
are the parts of the UML1.4 definition that are required to define SPEM models. These parts are defined exactly as in UML 1.4 section 2.5, except that some of the classes have been made abstract. There are also four small variations as follows:

- In Relationships (Figure 3-5) the connection end of the association between Association and AssociationEnd has multiplicity 2, instead of the 2..* specified by UML 1.4. This is because only binary associations are supported by SPEM.

- In Dependencies (Figure 3-6) the supplier and client associations between Dependency and ModelElement have multiplicity 1, instead of the 1..* specified by UML 1.4. This is because only binary dependencies are supported by SPEM.

- Every Feature has exactly 1 owner, instead of the 0..1 specified by UML.

- SPEM Associations are not Generalizable.

Figure 3-4 - Foundation Core Package - Backbone
Figure 3-5 - Foundation Core Package - Relationships

Figure 3-6 - Foundation Core Package - Dependencies
3.3 **SPEM/Foundation::Model_Management**

The SPEM/Foundation::Model_Management package is a subset of the UML 1.4 Model_Management package, and is shown in Figure 3-8. The elements in this package are defined exactly as in UML 1.4 section 2.14. Note that there is no ElementImport metaclass, used in UML to reify the concepts of aliasing and visibility; in SPEM there is no concept of visibility – all elements have public visibility - and elements imported into packages cannot be renamed.
3.4 SPEM_Foundation Well-Formedness Rules

The following well-formedness rules as found and numbered in the UML 1.4 specification apply to the SPEM_Foundation package.

Namespace:
[1] If a contained element, which is not an Association or Generalization has a name, then the name must be unique in the Namespace.
[2] All Associations must have a unique combination of name and associated Classifiers in the Namespace.

GeneralizableElement:
[3] Circular inheritance is not allowed.
[5] A GeneralizableElement may only be a child of GeneralizableElement of the same kind.

Constraint:

Classifier:
[3] No opposite AssociationEnds may have the same name in a Classifier

BehavioralFeature:
[1] All Parameters should have a unique name
[2] The type of the Parameters should be included in the namespace of the Classifier

AssociationEnd:
[2] An Instance may not belong by composition to more than one composite Instance.

Association:
[1] The AssociationEnds must have a unique name within the Association.
[2] At most one AssociationEnd may be an aggregation or composition.
4 Conceptual Model

At the core of the Software Process Engineering Metamodel (SPEM) is the idea that a software development process is a collaboration between abstract active entities called *process roles* that perform operations called *activities* on concrete, tangible entities called *work products* [20]. Figure 4-1 depicts this fundamental conceptual model using the UML notation for a class. Figure 4-1 and Figure 4-2 are not part of the proposed model and are given solely for explanatory reasons. They are intentionally very incomplete.

![Figure 4-1 - Conceptual Model](image)

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

From this model, a first step consists of “reifying” role, activity, and work product. This leads to the simple model shown in and Figure 4-2.

![Figure 4-2 - Reifying the conceptual model: roles, workproducts, and activities](image)
Chapter 3 explained how SPEM is built from the SPEM_Foundation package, which is a subset of UML 1.4, and the SPEM_Extensions package, which adds the constructs and semantics required for software process engineering.

Figure 5-1 shows the internal structure of the SPEM_Extensions package, in terms of its sub-packages, and also shows the dependencies between these packages and the SPEM_Foundations packages. We address each of the SPEM_Extensions subpackages in turn in the next five chapters: Basic Elements, Dependencies, Process Structure, Process Components and Process Lifecycle.

Figure 5-1 - SPEM Package Structure
6 Basic Elements
This package, detailed in Figure 6-1, defines the basic elements used for process description.

6.1 ExternalDescription
With every ModelElement is associated one or more ExternalDescriptions, which contain a description of the ModelElement suitable for a reader of the process description. ExternalDescriptions comprise the user-visible surface of the Software Process Description.

An ExternalDescription has four attributes of type String:
- content: a natural language description of the ModelElement
- name: the name of the ModelElement in a natural language
- language: the name of the natural language used for the value of content and name
- medium: a description of the medium and format of the ExternalDescription

![Diagram of Basic Elements package]

Figure 6-1 - Basic Elements package

6.2 Guidance
Zero or more Guidance elements may be associated with each ModelElement, to provide more detailed information to practitioners about the associated ModelElement.
Possible types of Guidance depend on the process family and can be for example: Guidelines, Techniques, Metrics, Examples, UML Profiles, Tool mentors, Checklist, Templates.

SPERM is designed to be flexible about the kinds of Guidance used in a process model, by reifying GuidanceKind as a separate class in the metamodel. Every Guidance is associated with a GuidanceKind, and the name of the GuidanceKind indicates what kind of Guidance it is. The following list of kinds of Guidance provides a basic repertoire; processes based on SPERM may add new kinds if required.

**Kinds of Guidance**

*Technique* is a kind of Guidance. A Technique is a detailed, precise “algorithm” used to create a work product. Techniques help to define the skills required to perform specific types of activities. The OPEN process uses the term ‘technique’. Other processes use ‘procedure’ or ‘directive’.

*UMLProfile* is a kind 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.

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

Figure 6-2 presents a diagram example of such an approach, where activities are connected to UML profiles. In this example, we see connections from ProcessRole occurrences such as “Analyst” as performers, to Activity occurrences such as “Elaborate Analysis”, and from Activity occurrences to a UMLProfile occurrence such as “UML analysis”.

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

*ToolMentor* is a kind of Guidance. A ToolMentor shows how to use a specific tool to accomplish an activity. Each ToolMentor is associated with a single Tool and inherits the association with the Activity it supports from Guidance. For example, “Using Rational ClearCase to Check Out and Check In Configuration Items” is a tool mentor in the RUP.
Figure 6-2 - Example of a process connecting activities to UML profiles

Guideline is a kind of Guidance. A Guideline is a set of rules and recommendations on how a given work product must look or must be organized. For example, 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.

Template is a kind of Guidance. A Template is a predefined document that provides a standardized format for a particular kind of WorkProduct. For example, “Microsoft Word template for Business Use Case Modeling”

Estimate is a kind 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.

QuadCycle defines also *Technology Roadmaps*: an explicit directive for technology use in the implementation of architectural styles, patterns and frameworks within the Global Industries Technology Architecture (GITA), and *Tacit Knowledge*: the experience and expertise of senior architects represented as a knowledge map in the Unisys Knowledge Management Initiative.
7 Dependencies

7.1 SPEM Dependencies

Figure 7-1 shows the Dependencies defined in SPEM. They are defined as subclasses of the SPEM_Foundation Dependency classes Abstraction, Usage and Permission, which have the semantics defined for UML 1.4\(^1\).

![Dependency Diagram]

Figure 7-1 - Dependencies

The following dependencies are supported by SPEM for process engineering:

- **Categorizes.** A Categorizes dependency acts from a Package to an individual process element in another package, and provides a means to associate process elements with multiple categories. This feature is both generally useful, and in particular acts in conjunction with Discipline (see section 9.4) to provide a top-level categorization of all elements.

- **Impacts.** An Impacts dependency acts from one WorkProduct to another WorkProduct to indicate that the modification of a WorkProduct could invalidate another.

---

\(^1\) In UML, specific types of Dependency are defined using stereotypes. In stand-alone SPEM, stereotypes are not available, so they are defined using subclasses.
For example, an important document in IBM’s Global Services Method is the Work Product Dependency diagram, represented in Figure 7-2. The icons in this diagram indicate Work Product Descriptions—in SPEM terms, instances of WorkProduct as described in section 8.1. The arrows represent instances of the *Impacts* Dependency in the IBM Global Services Method.

![Work Product Dependency diagram from IBM's GS Method](image)

**Figure 7-2 - Work Product Dependency diagram from IBM's GS Method**

- **Import.** An *Import* dependency denotes that the contents of the target Package are added to the namespace of the source Package. This has the same semantics as UML *Import* except that in SPEM all elements have public visibility.

- **Precedes.** A Precedes dependency acts from one Activity to another, or one WorkDefinition to another, to indicate finish-start or finish-finish dependencies between the work described, depending on the value of the *kind* attribute.

- **RefersTo.** A *RefersTo* dependency acts from one process element to another, to ensure that they are included in the same ProcessComponent, see section 9.2. The normal situation where this applies is where the text
of one process element refers, by name or content, to another element. In order to ensure consistency of meaning of the text, a *RefersTo* dependency should be established to give an explicit structural representation of such a dependency, so that when the referring element is included in a ProcessComponent, the referred-to element must also be included.

- **Trace.** A Trace dependency acts between WorkDefinitions or InformationElements and is mainly used to trace requirements and changes across models. It has the same semantics as UML *Trace*.

### 7.2 Well-formedness rules

**Categorizes:**

[1] The client must be a kind of Package

```ocl
categorizes inv:
  self.client.oclIsKindOf(Package)
```

**Impacts:**

[1] The supplier and client must be kinds of WorkProduct

```ocl
impacts inv:
  self.supplier.oclIsKindOf(WorkProduct) and self.client.oclIsKindOf(WorkProduct)
```

**Import:**

[1] The supplier and client must be kinds of Package

```ocl
import inv:
  self.supplier.oclIsKindOf(Package) and self.client.oclIsKindOf(Package)
```

**Precedes:**

[1] The supplier and client must be kinds of WorkDefinition

```ocl
precedes inv:
  self.supplier.oclIsKindOf(WorkDefinition) and self.client.oclIsKindOf(WorkDefinition)
```

**RefersTo:**

No additional rules.

**Trace:**

No additional rules.
8 Process Structure
This package, shown in Figure 8-1, defines the main structural elements from which a process description is constructed.

8.1 WorkProduct and InformationElement
A work product or 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. A WorkProduct describes one kind of work product. WorkProducts may consist of InformationElements. InformationElements are the lowest-level components of WorkProducts. They define the structure to be used for recording and presenting the various elements of information required during a project.

Associations
- WorkProduct and InformationElement are both specializations of Classifier. Thus they can participate in associations and contain nested definitions. They do not possess Features.
- A Workproduct description can describe WorkProducts that are aggregates of other WorkProducts. For example a software development plan (à la MIL-STD-498) consists of several other plans: Staffing plan, Configuration management plan, etc. This can represented using normal UML aggregation.

Figure 8-1 - Process Structure package
Attributes
The isDeliverable attribute on WorkProduct is true if that WorkProduct is defined as a formal deliverable of the process.

Note
Deliverable is not a major model element in SPEM because not all WorkProducts are deliverable, and whether a WorkProduct is delivered or not may change during the enactment.

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

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

8.2 WorkDefinition and ActivityParameter
WorkDefinition is a kind of BehaviouralFeature that describes the work performed in the process. Its main subclass is Activity, but Phase, Iteration, and Lifecycle (in the Process Lifecycle package) are also subclasses of WorkDefinition. WorkDefinition is not an abstract class, and instances of WorkDefinition itself can be created to represent composite pieces of work that are further decomposed. It has explicit inputs and outputs referred to via ActivityParameter.

Associations
- A WorkDefinitions can be composed of other WorkDefinitions using the association called subWork.
- A WorkDefinition is related to the WorkProducts it uses through the ActivityParameter class, which specifies whether they are used as input or output. The work described in the WorkDefinition uses the input workproducts, and creates or updates the output workproducts.
- A WorkDefinition has an owner ProcessPerformer, representing the primary role that performs that WorkDefinition in the process. In the case of Activities carried out by an individual or small group, this will be a ProcessRole. In the case of higher-level WorkDefinitions this will often be a single instance of ProcessPerformer that corresponds to the complete Process.
Attributes
The attribute *kind* on Parameter is used to indicate whether the associated work product is an input, output, a modifiable input, or a returned value to the WorkDefinition.

The attribute *hasWorkPerArtifact* indicates that multiple instances of the WorkDefinition are needed, one per instance of the corresponding WorkProduct. 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). This attribute can be true for at most one ActivityParameter per WorkDefinition.

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

1. Decomposition using subWork provides the means to describe that one WorkDefinition is composed of another and, therefore, the hierarchical nature of the WBS. When SPEM is represented as a UML Profile, subwork can be considered as an abstraction for the inclusion of the subsidiary WorkDefinitions on activity graphs, as explained in chapter 12.

2. The Precedes dependency provides the ability to sequence between elements of the WBS at the same level, see chapter 7.

Example
In the Fujitsu SDEM21 development process, there are 3 levels of WorkDefinition layers, the last of which corresponds to activities.

8.3 Activity and Step
*Activity* is the main subclass of WorkDefinition. It describes a piece of work performed by one ProcessRole: the tasks, operations, and actions that are performed by a role or with which the role may assist. An Activity may consist of atomic elements called *Steps.*

Associations
- Activity inherits from WorkDefinition the fact that it has input and output parameters, of type WorkProduct.
- An Activity is owned by a ProcessRole that is the performer (or owner) of the described activity. It may refer to additional ProcessRoles that are the assistants in the activity by including these as additional input parameters to the Activity.
- Although this is not explicitly prohibited, an Activity does not normally use the subWork structure inherited from WorkDefinition; instead decomposition
within Activity is done using Steps. A Step is described in the context of the enclosing Activity in terms of the ProcessRoles and WorkProducts it uses.

Examples

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

In IBM’s Global Services Method, *Specify Solution Requirements* is an example of a WorkDefinition. It is decomposed into several “tasks”, modeled by SPEM’s Activity, such as *Detail Usability Requirements.*

Synonyms

The Rational Unified Process and QuadCycle use ‘activity’ composed of a partially ordered set of ‘steps’. The IBM process defines ‘activities’ that corresponds to SPEM WorkDefinition, consisting of ‘tasks’ and ‘subtasks’ that corresponds to SPEM Activities. OPEN uses ‘task’.

8.4 **ProcessPerformer and ProcessRole**

A *ProcessPerformer* defines an owner for a set of WorkDefinitions in a process. ProcessPerformer has a subclass, ProcessRole. ProcessPerformer represents abstractly the “whole process” or one of its components, and is used to own WorkDefinitions that do not have a more specific owner. ProcessRole defines responsibilities over specific WorkProducts, and defines the roles that perform and assist in specific activities.

associations

- ProcessPerformer is a specialization of Classifier, and thus may participate in inheritance relationships and associations within the process definition.
- A ProcessRole is responsible for a set of WorkProducts; this is modeled by creating M1-level associations between the ProcessRole and the relevant WorkProducts.
- A ProcessRole is the owner (performer) of Activities.
- A ProcessPerformer is the owner of higher level aggregate WorkDefinitions that cannot be associated with individual ProcessRoles.

Synonyms

ProcessRole is called ‘role’ in the IBM Global Services Method, DMR Macroscope and in OPEN [4], and it was called ‘worker’ in the Rational Unified Process [1, 3], prior to RUP 2001. We have also encountered ‘agent’.
Examples
In the Rational Unified Process, examples of ProcessRole are Architect, Analyst, Technical Writer, and Project Manager to name a few.

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

8.5 Well-formedness rules

Activity
[1] Each Activity is imported by exactly one Discipline.
context Activity inv:
   self.supplierDependency.select (d | d.oclIsKindOf(Import)).client.select (c | c.oclIsKindOf(Discipline))->size = 1

[2] Every Activity is owned by a ProcessRole.
context Activity inv:
   self.owner.oclIsKindOf(ProcessRole)

ActivityParameter
No additional rules.

InformationElement
No additional rules.

ProcessPerformer
[1] Every feature must be a kind of WorkDefinition.
context ProcessPerformer inv:
   self.feature->forall(f | f.oclIsKindOf(WorkDefinition))

ProcessRole
[1] Every feature must be a kind of Activity.
context ProcessRole inv:
   self.feature->forall(f | f.oclIsKindOf(Activity))

Step
No additional rules.

WorkDefinition
[1] A WorkDefinition is owned by a kind of ProcessPerformer.
context WorkDefinition inv:
   self.owner.oclIsKindOf(ProcessPerformer)
WorkProduct
No additional rules.
9 Process Components

Figure 9-1 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 Package

Just as in UML, a Package is a container that can both own and import process definition elements. Activities and WorkDefinitions are owned, respectively, by ProcessRoles and ProcessPerformers; other SPEM ModelElements can be owned by Packages.

Packages and the Categorizes dependency can be used to implement general categorization of process description elements. A Package is created to represent each category, and all of the elements linked via a Categorizes dependency into that Package to represent membership of the category. Multiple overlapping categories can be created to serve various purposes in process engineering. A more specific kind of categorization of Activities is implemented by Discipline, see section 9.4.

9.2 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 imports a non-arbitrary set of process definition elements, modeled in SPEM by ModelElements. Such a set must be self-contained; this means that there are no RefersTo dependencies 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.

Example

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 WorkProducts 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, ProcessRoles, 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 SPEM permits both of these kinds of composition but provides no explicit support for either.

![Figure 9-1 - Process Components package](image)

9.3 Process

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.

A Lifecycle, as defined in section 10.1 is associated with 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.
9.4 Discipline

A *Discipline* is a particular specialization of Package that partitions the Activities within a process according to a common “theme”. Partitioning the Activities in this way implies that the associated Guidance and output WorkProducts are similarly categorized under the theme. The inclusion of an Activity in a Discipline is represented by the Categorizes dependency, with the additional constraint that every Activity is categorized by exactly one Discipline.

Example


Synonyms

The IBM processes use the term ‘domain’; the Rational Unified Process uses ‘core workflow’; the Fujitsu SDEM21 uses ‘category’; Objectory used ‘process component’; Fusion uses the term ‘phase’, OPEN uses the work ‘activity’.

9.5 Well-formedness rules

ProcessComponent

A process component must be self-contained, i.e. there are no links (associations or dependencies) to anything outside the component.

[1] No dependencies outside the component.

```plaintext
context ProcessComponent inv:
let includedElements : Set(ModelElement) =
self.clientDependency->select
(d | d.oclIsKindOf(Import)).supplier in
includedElements->forall ( e | e.clientDependency.supplier->forall ( m | includedElements->includes(m))) and
includedElements->forall ( e | e.supplierDependency.client->forall ( m | includedElements->includes(m)))
```

[2] No associations outside the component.

```plaintext
context ProcessComponent inv:
let includedElements : Set(ModelElement) =
```
self.clientDependency->select
(d | d.oclIsKindOf(Import)).supplier in

includedElements->forall ( e |
  e.allAssociatedInstances->forall ( i |
    includedElements -> includes(i))

where allAssociatedInstances cannot easily be defined in OCL, but could be defined by slightly extending OCL as follows:

i.allAssociatedInstances =
  i.type.associationEnds->collect(ae |
    i.navigate(ae))

Process
No additional rules.

Discipline
[1] Disciplines only categorize Activities.
context Discipline inv:
  self.clientDependency->select(d |
    d.oclIsKindOf(Categorizes)).supplier->forall(m |
    m.oclIsKindOf(Activity))
10 Process Lifecycle

In this package, shown in Figure 10-1, we introduce 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 of the process description that are used to help plan and execute enactments of that description.

![Figure 10-1 - Process Lifecycle package](image)

10.1 Phase

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

10.2 Lifecycle
A process *Lifecycle* is defined as a sequence of Phases that achieve a specific goal. It defines the behavior of a complete process to be enacted in a given project or program.

Associations
A Lifecycle is associated with a sequence of Phases by the use of the subWork association, see section 8.2.

A Lifecycle is associated with one or more Processes via the governedProcesses association that associates a Lifecycle (describing the behavior of the process) with a Process (that packages up all of the descriptive material contained in the process).

Example
The DMR Macroscope describes 3 system delivery lifecycles: a *Generic Development* path, an *Accelerated Development* path, and a *Package Solution Delivery* path. The Fujitsu SDEM21 provides a specific lifecycle for component-based development called *ComponentAA*.

10.3 Iteration
An Iteration is a composite WorkDefinition with a minor milestone.

Example
The following example work breakdown structure showing Iterations is from the DMR Macroscope:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Iteration</th>
<th>Activity</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preliminary Analysis</td>
<td>First Joint Requirements Planning (JRP) Workshop</td>
<td>Define Owner Requirements</td>
<td>Define objectives based on stated needs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
10.4 Precondition and Goal

With each WorkDefinition can be associated a Precondition and a Goal. Preconditions and Goals are Constraints, where the constraint is expressed in the form of a BooleanExpression (which is a string) following syntax similar to that of a guard condition in UML. The condition is expressed in terms of the states of the WorkProducts that are the parameters of the WorkDefinition or of an enclosing WorkDefinition.
Example
If a WorkDefinition called DesignReview has input parameters DesignModel and DesignStandards and output parameter ReviewActions, then a Precondition can have the form
   (DesignModel in state Ready) and (DesignStandards in state Approved)
and a Goal
   (ReviewActions in state Drafted)

10.5 Well-formedness rules

Goal
No additional rules.

Iteration
No additional rules.

Lifecycle
[1] Lifecycles only contain Phases.
context Lifecycle inv:
   self.subWork->forall(ph | ph.oclIsKindOf(Phase))

Phase
No additional rules.

Precondition
No additional rules.
11 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 SPEM. See standards IEEE 610.12-1990 or ISO 12207.

All SPEM Elements (modeled as ModelElements) 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 SPEM Element. 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 Process Definition Elements from a consistent set of version histories all belonging to the same variant.
12 SPEM as a UML Profile

In the chapters so far, SPEM has been directly defined as a metamodel. SPEM can be used by directly instantiating this stand-alone metamodel. But SPEM is also defined as a UML Profile.

SPEM is dedicated to software processes modeling. Many features of the UML provide the necessary basis for modeling processes, and many other UML features provide useful additional modeling capacities. Being a UML profile, SPEM both defines modeling capacities dedicated to the software process domain, and gains the benefit of the expressiveness of UML. For example, Use Case modeling, which is sometimes used for modeling processes, is not defined as a specific SPEM facility, but can be inherited from UML.

Alignment with various process modeling languages is another advantage of using UML Profiles. The SPEM profile uses extensively the UML Activity Diagram model to give more detail to the work decomposition that is represented in the stand-alone metamodel by the WorkDefinition::subWork association. It is expected that the various kind of process modeling techniques (Business Process Modeling, Workflow, etc.) will be aligned with UML at some time. SPEM will naturally benefit from this convergence, and from any other convergence and improvements that will occur with UML.

Finally, a wide community of software developers is familiar with UML and uses a UML case tool environment. Defining a UML profile allows this important community to reuse its modeling knowledge and tools in the software-process modeling domain.

The UML 1.4 definition of profile is given in section 1.5 of this document, and is repeated here:

A profile stereotype of Package contains one or more related extensions of standard UML semantics (refer to Section 2.6, “Extension Mechanisms”). These are normally intended to customize UML for a particular domain or purpose. Profiles can contain stereotypes, tag definitions, and constraints. They can also contain data types that are used by tag definitions for informally declaring the types of the values that can be associated with tag definitions.

In addition, a profile package can specify a related model library and identify a subset of the UML metamodel that is applicable for the profile. In principle, profiles merely refine the standard semantics of UML by adding further
constraints and interpretations that capture domain-specific semantics and modeling patterns. They do not add any new fundamental concepts.

In order to define a UML profile for SPEM, the following must be done.

1. Identify that subset of the UML metamodel classes to be included in the profile.

2. For most classes in the SPEM metamodel, identify a “base class” in the UML metamodel subset that will, when stereotyped appropriately, act in place of the SPEM class. The technique used here is specified in section 3.35.2 of the UML 1.4 specification. The fact that SPEM is itself defined as an extension of a subset of UML makes this very straightforward. For the one class (GuidanceKind) in the SPEM metamodel to which the base class technique does not apply, the semantics of instances of that class are emulated using UML stereotypes.

3. For each attribute and association in the SPEM metamodel, define a way to emulate that attribute or association. In the SPEM profile, attributes are emulated by means of TaggedValues. Most associations have close analogues in the UML metamodel. Those that don’t, get special treatment as detailed below.

4. For those parts of the UML subset that have a plausible mapping into SPEM concepts, but are not used directly to emulate the SPEM metamodel, show how they are mapped into SPEM-related concepts. For SPEM, this applies particularly to the use of Use Case diagrams, Activity diagrams and state machines.

5. Give additional constraints over the UML metamodel that are implied by the use of the profile.

6. Define notational icons for SPEM concepts that are represented by UML stereotypes.

The remaining parts of this chapter deal with each of these topics.

12.1 Identified subset of the UML Metamodel

The SPEM profile retains the following packages from the UML Metamodel:

- Core
except Method (from Backbone)
except Binding (from Dependencies)
except Node, Interface, Artifact and Component (from Classifiers)
except TemplateParameter and TemplateArgument (from AuxiliaryElements)

- ExtensionMechanisms
- DataTypes
- CommonBehavior
  except ComponentInstance, NodeInstance
- Collaboration
- UseCases
- StateMachines
- ActivityGraphs
- ModelManagement
  except Subsystem

All of the classes in the SPEM_Foundation package (see chapter 3) together with their attributes and associations, are directly represented by the equivalent UML classes, attributes and associations.

The following table shows which UML base element is used for each class of the SPEM_Extensions package (chapters 6 - 10).

<table>
<thead>
<tr>
<th>SPEM Metaclass</th>
<th>UML Base element</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guidance</td>
<td>Core::Comment</td>
<td></td>
</tr>
<tr>
<td>GuidanceKind</td>
<td>N/A</td>
<td>Instances are emulated by UML stereotypes</td>
</tr>
<tr>
<td>ExternalDescription</td>
<td>Core::PresentationElement</td>
<td></td>
</tr>
<tr>
<td>Trace</td>
<td>Core::Dependency</td>
<td></td>
</tr>
<tr>
<td>RefersTo</td>
<td>Core::Dependency</td>
<td></td>
</tr>
<tr>
<td>Precedes</td>
<td>Core::Dependency</td>
<td></td>
</tr>
<tr>
<td>Impacts</td>
<td>Core::Dependency</td>
<td></td>
</tr>
<tr>
<td>Categorizes</td>
<td>Core::Dependency</td>
<td></td>
</tr>
<tr>
<td>Import</td>
<td>Core::Dependency</td>
<td></td>
</tr>
<tr>
<td>WorkDefinition</td>
<td>Core::Operation</td>
<td></td>
</tr>
<tr>
<td>Activity</td>
<td>Core::Operation</td>
<td></td>
</tr>
<tr>
<td>ActivityParameter</td>
<td>Core::Parameter</td>
<td></td>
</tr>
<tr>
<td>WorkProduct</td>
<td>Core::Class</td>
<td></td>
</tr>
<tr>
<td>InformationElement</td>
<td>Core::Class</td>
<td></td>
</tr>
<tr>
<td>ProcessPerformer</td>
<td>UseCases::Actor</td>
<td></td>
</tr>
<tr>
<td>ProcessRole</td>
<td>UseCases::Actor</td>
<td></td>
</tr>
<tr>
<td>Step</td>
<td>ActivityGraphs::ActionState</td>
<td>See Figure 12-3</td>
</tr>
</tbody>
</table>
### Notes:

- Activity and its superclass WorkDefinition are based on UML Operation; this allows the use of ActivityGraphs at every level of process description, from a diagram showing phases of a lifecycle, to the details of the steps of an activity description.
- Instances of GuidanceKind, such as Techique, UMLProfile, ToolMentor etc (see section 6.2) are represented in the profile as stereotypes of Guidance.
- WorkProduct is a stereotype of UML Class. Aggregation and association of WorkProduct descriptions can use the normal UML aggregation and association.
- The decomposition of WorkDefinition is shown in Figure 12-3 below.

### 12.2 Mapping to UML base classes

Most mappings are very simple, see Figure 12-2 as they follow the pattern shown in Figure 12-1.

![Figure 12-1 - Pattern for most classes, from a SPEMClass to the UML Base Class it maps to.](image-url)
12.3 Attributes

Attributes in the SPEM_Extensions package are represented by TaggedValues, as shown in the following table.

All tag definitions have the multiplicity 1.

<table>
<thead>
<tr>
<th>TagDefinition</th>
<th>Type</th>
<th>on stereotype</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasWorkPerArtifact</td>
<td>Boolean</td>
<td>ActivityParameter</td>
<td>When true, the WorkDefinition will be enacted once for every instance of the corresponding WorkProduct</td>
</tr>
<tr>
<td>content</td>
<td>String</td>
<td>ExternalDescription</td>
<td>Description of the annotated model element</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>ExternalDescription</td>
<td>Name of the external description</td>
</tr>
<tr>
<td>medium</td>
<td>String</td>
<td>ExternalDescription</td>
<td>Medium of the external description eg textual, audio, graphics, etc</td>
</tr>
<tr>
<td>language</td>
<td>String</td>
<td>ExternalDescription</td>
<td>Language, such as English, French, Japanese, in which the description is provided</td>
</tr>
<tr>
<td>kind</td>
<td>[s_s, f_s]</td>
<td>Precedes</td>
<td>Which kind of precedence dependency is being described</td>
</tr>
<tr>
<td>isDeliverable</td>
<td>Boolean</td>
<td>WorkProduct</td>
<td>True when the work product is defined as a formal deliverable of the process</td>
</tr>
</tbody>
</table>
12.4 Associations

Associations in the SPEM_Extensions package are represented in a variety of ways, as follows.

**Guidance::kind.** This is not required, because instances of GuidanceKind are represented as stereotypes of Guidance.

**Guidance::annotatedElement.** This is represented by the UML association Comment::annotatedElement.

**ActivityParameter::type.** This is represented by the UML association Parameter::type.

**WorkDefinition::owner.** This is represented by the UML association Feature::owner.

**WorkDefinition::subWork.** This is not represented directly. Instead it is represented using ActivityGraphs, as shown in Figure 12-3. In green (dark) are the UML base classes that together correspond to the subWork association: ActivityGraph, CompositeState, ActionState, CallAction.

**Activity::steps.** This is also represented by ActivityGraphs, as shown in Figure 12-3.

**WorkDefinition::goal** and **WorkDefinition::precondition.** These are represented by the UML association ModelElement::constraint.

**Process::governingLifecycle.** This is represented by a new stereotype of Abstraction called «governs», which acts between a Lifecycle and the processes that it is related to.
12.5 Use of Activity diagrams and Use Case diagrams.

It has already been noted that Activity Graphs are used in the UML profile version of SPEM to give a more detailed decomposition of WorkDefinition. In chapter 13 this document defines a set of icons for use in process definitions. In particular, there are particular icons used to represent the classes WorkProduct, Activity, and WorkDefinition.

In SPEM, these icons may appear uniformly on all UML diagrams in which these concepts are referred to. However, in the case of Activity diagrams, these elements are not referred to directly. Instead, instances of ActionState appear, which may be thought of as “notational proxies” for corresponding instances of WorkDefinition and Activity. Similarly, instances of ObjectFlowState act as proxies for corresponding instances of WorkProduct.

To resolve this issue, the SPEM profile allows ActionState to appear as an alternative base class for the stereotypes Activity and WorkDefinition. In both cases, the idea is that the notational element is a proxy for the stereotyped Operation associated with the CallAction of the ActionState. Similarly, the profile allows ObjectFlowState to appear as an alternative base class for
WorkProduct, with the interpretation that the notational element is a proxy for the stereotyped Classifier associated with the ObjectFlowState.

A similar issue arises because SPEM uses Use Case diagrams to illustrate the relationships between ProcessRole/ProcessPerformer and Activity/WorkDefinition. To enable this the profile allows UseCase to be a further alternative base class for WorkDefinition and Activity. To complete this interpretation, a UML «realize» dependency should be created between the WorkDefinition orActivity and the Use Case that it represents.

### 12.6 Use of State diagrams

UML state diagrams may be used to define the states of a WorkProduct. The resulting state definitions may be used to constrain the valid expressions that can be used in a SPEM Precondition or Goal.

### 12.7 Stereotypes of the SPEM profile

The following table gives a complete summary of all of the SPEM profile stereotypes, based on the discussion above.

Note that the following stereotypes are added for notational convenience: ProcessPackage (special notation for packages in a SPEM context), Document and Model (special notation for different kinds of WorkProduct). Apart from the icons and their implied connotations, these stereotypes have no additional semantics above those of their base classes.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Stereotype Parent</th>
<th>Comment</th>
<th>Constraints (see below)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WorkProduct</td>
<td>Class</td>
<td>ObjectFlowState</td>
<td>See § 8.1</td>
<td></td>
</tr>
<tr>
<td>ActivityParameter</td>
<td>Parameter</td>
<td></td>
<td>See § 8.2</td>
<td></td>
</tr>
<tr>
<td>Goal</td>
<td>Constraint</td>
<td>postcondition</td>
<td>See § 10.4</td>
<td></td>
</tr>
<tr>
<td>Precondition</td>
<td>Constraint</td>
<td>precondition</td>
<td>See § 10.4</td>
<td></td>
</tr>
<tr>
<td>WorkDefinition</td>
<td>Operation</td>
<td>ActionState</td>
<td>See § 8.2</td>
<td>P2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UseCase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step</td>
<td>ActionState</td>
<td></td>
<td>See § 8.3</td>
<td></td>
</tr>
<tr>
<td>Guidance</td>
<td>Comment</td>
<td></td>
<td>See § 6.2</td>
<td></td>
</tr>
<tr>
<td>ExternalDescription</td>
<td>PresentationElement</td>
<td></td>
<td>See § 6.1</td>
<td></td>
</tr>
<tr>
<td>Activity</td>
<td>Operation</td>
<td>ActionState</td>
<td>See § 8.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UseCase</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
12.8 Well-formedness rules

In translating the stand-alone model to a UML profile, there are various sources of additional or changed well-formedness rules.
Restricted multiplicities
As pointed out in chapter 3, the stand-alone SPEM metamodel is based on a subset of UML with some restrictions on the multiplicities. These restrictions also apply to the UML profile.

Use of context and oclIsKindOf
In the presence of stereotypes, the use of oclIsKindOf needs in principle to be modified. We assume that the meaning of oclIsKindOf can be extended in the presence of stereotypes, so that if oclIsKindOf refers to a stereotype name, it delivers true if the tested element has that stereotype or a sub-stereotype.

Similarly, constraints on stereotypes are written under the assumption that it is valid to use a stereotype name in the context part of the constraint. Strictly-speaking this is a shorthand, for example:

```
context ProcessPackage inv: X
```

can be considered as a shorthand for

```
context Package inv:
  (self.stereotype.name = “ProcessComponent” or
  self.stereotype.name = “Process” or
  self.stereotype.name = “Discipline”) implies X
```

With these provisos, all of the well-formedness rules in earlier chapters apply to the profile.

Profile-specific rules
The following rules apply to the construction of the profile itself.

governs
[P1] A governs dependency acts between a Lifecycle and a ProcessPerformer

```
context Dependency inv:
  self.stereotype.name = “governs” implies
    self.supplier->exists(stereotype.name=“Lifecycle”)
    and
    self.client->exists(stereotype.name=“ProcessPerformer”)
```

WorkDefinition
[P2] A WorkDefinition behavior is defined using no more than a single Activity Graph and in no other way.
context WorkDefinition inv:
  self.behavior->size <= 1
and
  self.behavior->forall( b | b.OCLIsTypeOf(ActivityGraph))

ActionState

[P3] An ActionState is either a Step or refers to a CallAction for another WorkDefinition:

context ActionState inv:
  self.stereotype.name = "Step" or
  (self.entry->size = 1 and
   self.entry.oclIsKindOf(CallAction) and
   self.entry.operation.oclIsKindOf(WorkDefinition))
13 Notation

13.1 Diagrams
Basic UML diagrams can be used to present different perspectives of a software process model. In particular, the following UML notations are useful:
- Class diagram
- Package diagram
- Activity diagram
- Use case diagram
- Sequence diagram
- Statechart diagram

Because some semantic elements of UML have been excluded from SPEM, the following notations should not be used:
- implementation diagrams
- component or node diagrams.

There are some notation and diagrams that are not excluded, but for which we have not specified any mapping nor meaning.

13.2 Suggested icons
Column "Notation" in table "Stereotypes" in section 12.7 suggests alternate representations for most frequently used concrete classes of the metamodel. These icons can be used in modeling a software development process to represent activities, work products, process roles, etc. It is suggested to replace the regular symbol with these icons as shown in the examples below.

13.3 Class diagram
Class diagrams allow the representation of the following aspects of a software process:
- Inheritance
- Dependencies
- Simple associations
- Comments to point to the guidance (for example URL link)
- Relations between ProcessPerformer or ProcessRole and WorkProduct
- Structure, decomposition and dependencies of WorkProducts (see example in Figure 13-1)
However, some restrictions apply when using class diagrams in conjunction with SPEM. More specifically, the following notational elements should not be used:
- Interface
- Template
- White diamond
- Qualified associations
- N-ary associations

![Figure 13-1 - Example of Class Diagram](image)

13.4 **Package diagram**

Package diagrams allow the representation of Process, ProcessComponents, ProcessPackages and Disciplines. Nested and non-nested forms can be used, but subsystems should not appear in such diagrams.

13.5 **Use case diagram**

Use case diagrams show the relationship between process roles and the main work definitions. No particular restrictions apply. See example in Figure 13-2.
13.6 Sequence diagrams

Sequence diagrams can be used to illustrate interaction patterns among SPEM model element instances. Only stick arrowheads should be used.

13.7 Statechart diagrams

Statechart diagrams can be used to illustrate the behaviour of SPEM model elements. Nesting and parallelism are allowed, but signal declaration and history indicators are not.

13.8 Activity diagrams

Activity diagrams allow presenting the sequencing of activities with their input and output work products as well as object flow states. Swimlanes can be used to separate the responsibilities of different process roles.
Figure 13-3 - Example of Activity diagram
14 XMI DTD For the SPEM

In the case where SPEM is represented as a MOF metamodel, rather than as a UML profile, a XMI DTD corresponding to that metamodel must be used to interchange SPEM models. Such a DTD is reproduced here.

Note that every package in the stand-alone definition of SPEM is stereotyped as «metamodel», and has the MOF Tag 'uml2mof.hasImplicitReferences' set to true. This means that references are automatically generated for all navigable association ends, and corresponding elements are generated in the XMI DTD.

<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- XMI is the top-level XML element for XMI transfer text           -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI (XMI.header?, XMI.content?, XMI.difference*,
XMI.extensions*)>
<!ATTLIST XMI
  xmi.version CDATA #FIXED "1.1"
  timestamp CDATA #IMPLIED
  verified (true|false) #IMPLIED>
<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- XMI.header contains documentation and identifies the model,      -->
<!-- metamodel, and metametamodel                                    -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.header (XMI.documentation?, XMI.model*, XMI.metamodel*,
XMI.metametamodel*, XMI.import*)>
<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- documentation for transfer data                                -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.documentation (#PCDATA | XMI.owner | XMI.contact |
XMI.longDescription | XMI.shortDescription |
XMI.exporter | XMI.exporterVersion |
XMI.notice)*>
<!ELEMENT XMI.owner ANY>
<!ELEMENT XMI.contact ANY>
<!ELEMENT XMI.longDescription ANY>
<!ELEMENT XMI.shortDescription ANY>
<!ELEMENT XMI.exporter ANY>
<!ELEMENT XMI.exporterVersion ANY>
<!ELEMENT XMI.exporterID ANY>
<!ELEMENT XMI.notice ANY>
<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- XMI.element.att defines the attributes that each XML element    -->
<!-- that corresponds to a metamodel class must have to conform to   -->
<!-- the XMI specification.                                         -->
<!-- _______________________________________________________________ -->
<!ENTITY % XMI.element.att '
xmi.id ID #IMPLIED xmi.label CDATA #IMPLIED xmi.uuid
CDATA #IMPLIED '>
<!-- _______________________________________________________________ -->
<!--                                                                 -->
<!-- XMI.link.att defines the attributes that each XML element that  -->
<!-- corresponds to a metamodel class must have to enable it to      -->
<!-- function as a simple XLink as well as refer to model constructs within the same XMI file. -->

<!ENTITY % XMI.link.att "href CDATA #IMPLIED xmi.idref IDREF #IMPLIED xml:link CDATA #IMPLIED xlink:inline (true|false) #IMPLIED xlink:actuate (show|user) #IMPLIED xlink:content-role CDATA #IMPLIED xlink:title CDATA #IMPLIED xlink:show (embed|replace|new) #IMPLIED xlink:behavior CDATA #IMPLIED'>

<!-- XMI.model identifies the model(s) being transferred -->

<!ELEMENT XMI.model ANY>
<!ATTLIST XMI.model %XMI.link.att;
 xmi.name    CDATA #REQUIRED
 xmi.version CDATA #IMPLIED>

<!-- XMI.metamodel identifies the metamodel(s) for the transferred data -->

<!ELEMENT XMI.metamodel ANY>
<!ATTLIST XMI.metamodel %XMI.link.att;
 xmi.name    CDATA #REQUIRED
 xmi.version CDATA #IMPLIED>

<!-- XMI.metametamodel identifies the metametamodel(s) for the transferred data -->

<!ELEMENT XMI.metametamodel ANY>
<!ATTLIST XMI.metametamodel %XMI.link.att;
 xmi.name    CDATA #REQUIRED
 xmi.version CDATA #IMPLIED>

<!-- XMI.import identifies imported metamodel(s) -->

<!ELEMENT XMI.import ANY>
<!ATTLIST XMI.import %XMI.link.att;
 xmi.name    CDATA #REQUIRED
 xmi.version CDATA #IMPLIED>

<!-- XMI.content is the actual data being transferred -->

<!ELEMENT XMI.content ANY>

<!-- XMI.extensions contains data to transfer that does not conform to the metamodel(s) in the header -->

<!ELEMENT XMI.extensions ANY>
<!ATTLIST XMI.extensions
 xmi.extender CDATA #REQUIRED>

<!-- extension contains information related to a specific model construct that is not defined in the metamodel(s) in the header -->
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.extension ANY>
<!ATTLIST XMI.extension %XMI.element.att; %XMI.link.att; 
xmi.extender CDATA #REQUIRED 
xmi.extenderID CDATA #IMPLIED>
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.difference (XMI.difference | XMI.delete | XMI.add | 
XMI.replace)*>
<!ATTLIST XMI.difference %XMI.element.att; %XMI.link.att;>
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.delete EMPTY>
<!ATTLIST XMI.delete %XMI.element.att; %XMI.link.att;>
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.add ANY>
<!ATTLIST XMI.add %XMI.element.att; %XMI.link.att; 
xmi.position CDATA "-1">
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.replace ANY>
<!ATTLIST XMI.replace %XMI.element.att; %XMI.link.att; 
xmi.position CDATA "-1">
<!-- _______________________________________________________________ -->
<!ELEMENT XMI.reference ANY>
<!ATTLIST XMI.reference %XMI.link.att;>
<!ATTLIST XMI xmlns:UML CDATA #IMPLIED>
<!-- ========= UML:Data_Types ========= -->
<!ENTITY % UML:AggregationKind '(none|aggregate|composite)'>
<!ENTITY % UML:ParameterDirectionKind '(in|inout|out|return)'>
<!-- ========= UML:Multiplicity ========= -->
<!ELEMENT UML:Multiplicity.range (UML:Multiplicity)*>
<!ENTITY % UML:MultiplicityFeatures 'XMI.extension | 
UML:Multiplicity.range'>
<!ENTITY % UML:MultiplicityAtts '%XMI.element.att; %XMI.link.att;'>
<!ENTITY UML:Multiplicity (%UML:MultiplicityFeatures;)*>
<!ATTLIST UML:Multiplicity %UML:MultiplicityAtts;>
<!-- ========= UML:MultiplicityRange ========= -->
<!ELEMENT UML:MultiplicityRange.multiplicity (UML:Multiplicity)*>
<!ENTITY % UML:MultiplicityRangeFeatures 'XMI.extension | 
UML:Multiplicity.range'>
<!ENTITY % UML:MultiplicityRangeAtts '%XMI.element.att; %XMI.link.att;'>
<!ELEMENT UML:MultiplicityRange (%UML:MultiplicityRangeFeatures;)*>
<!ATTLIST UML:MultiplicityRange %UML:MultiplicityRangeAtts;>
<!-- -------- UML:Categorizes -------- -->
<!ENTITY % UML:CategorizesFeatures 'UML:UsageFeatures;'>
<!ENTITY % UML:CategorizesAtts 'UML:UsageAtts;'>
<!ELEMENT UML:Categorizes (%UML:CategorizesFeatures;)*>
<!ATTLIST UML:Categorizes %UML:CategorizesAtts;>
15 Glossary

Activity
A Work Definition describing what a Process Role performs. Activities are the main element of work.

Component (see Process Component)

Dependency
A Dependency is a process-specific relationship between process Model Elements.

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

Element (see Model Element)

Guidance
Guidance is a Model Element associated with the major process definition elements, which contains additional descriptions such as techniques, guidelines and UML profiles, procedures, standards, templates of work products, examples of work products, definitions, and so on.

Iteration
An Iteration is a large-grained Work Definition that represents a set of Activities focusing on a portion of the system development that results in a release (internal or external) of the software product.

Model Element
An element describing one aspect of a software engineering process.

Process Role
A Model Element describing the roles, responsibilities and competencies of an individual carrying out Activities within a Process, and responsible for certain Work Products.

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

Process

Process Component
A Process Component is a coherent grouping of process Model Elements organized from a given vantage point such as a discipline, for example,
testing, or the production of some specific work product, for example, requirements management.

**Process Performer**
A Process Performer is a Model Element describing the owner of Work Definitions. Process Performer is used for Work Definitions that cannot be associated with individual Process Roles, such as a Life Cycle or a Phase.

**Step**
An atomic and fine-grained Model Element used to decompose Activities. Activities are partially ordered sets of Steps.

**Work Definition**
A Model Element of a process describing the execution, the operations performed, and the transformations enacted on the Work Products by the roles. Activity, Iteration, Phase, and Lifecycle are kinds of work definition.

**Work Product**
A Work Product is a description of a piece of information or physical entity produced or used by the activities of the software engineering process. Examples of work products include models, plans, code, executables, documents, databases, and so on.
16 References

This chapter 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.

27. Unisys QuadCycle: A full life cycle component based development and deployment methodology based on Rational Unified Process and Unisys TeamMethod
## Annex 1: Translation table

This annex maps the terminology from different sources:

<table>
<thead>
<tr>
<th>Source</th>
<th>Role</th>
<th>Activity</th>
<th>Step</th>
<th>Work Product</th>
<th>Information-Element</th>
<th>Discipline</th>
<th>Lifecycle</th>
<th>Phase</th>
<th>Iteration</th>
<th>Guidance</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEM</td>
<td>ProcessRole</td>
<td>Activity</td>
<td>Step</td>
<td>WorkProduct</td>
<td>Information-Element</td>
<td>Discipline</td>
<td>Lifecycle</td>
<td>Phase</td>
<td>Iteration</td>
<td></td>
</tr>
<tr>
<td>Rational Unified Process</td>
<td>Role</td>
<td>Activity</td>
<td>Step</td>
<td>Artifact</td>
<td>Discipline</td>
<td>Process</td>
<td>Phase</td>
<td></td>
<td></td>
<td>Guidelines</td>
</tr>
<tr>
<td>IBM Global Services Method</td>
<td>Role</td>
<td>Task</td>
<td></td>
<td>Work Product</td>
<td>Description</td>
<td>Domain</td>
<td>Engagement Model</td>
<td>Phase</td>
<td>Iteration</td>
<td>Technique</td>
</tr>
<tr>
<td>DMR Macroscope</td>
<td>Role</td>
<td>Activity</td>
<td></td>
<td>Delivery</td>
<td>Product</td>
<td>Domain</td>
<td>Path</td>
<td>Phase</td>
<td>Iteration</td>
<td>Guideline Technique</td>
</tr>
<tr>
<td>Unisys QuadCycle</td>
<td>Role</td>
<td>Activity</td>
<td>Step</td>
<td>Artifact</td>
<td>Asset</td>
<td>Discipline</td>
<td>Process</td>
<td>Phase</td>
<td>Iteration</td>
<td>Guideline Technique</td>
</tr>
<tr>
<td>OPEN</td>
<td>Role</td>
<td>Direct</td>
<td>producer</td>
<td>Task</td>
<td>Subtask</td>
<td>Work product</td>
<td>Activity</td>
<td>Lifecycle process</td>
<td>Phase</td>
<td></td>
</tr>
<tr>
<td>Fujitsu SDEM21</td>
<td>Role</td>
<td>WorkItem</td>
<td></td>
<td>Document</td>
<td>File</td>
<td>Category</td>
<td>Lifecycle process</td>
<td>Phase</td>
<td></td>
<td>Guidelines</td>
</tr>
<tr>
<td>OOSP</td>
<td>Task</td>
<td>Activity</td>
<td></td>
<td>Deliverable</td>
<td></td>
<td></td>
<td></td>
<td>Phase</td>
<td></td>
<td>Guideline Standard</td>
</tr>
<tr>
<td>Promoter</td>
<td>Role</td>
<td>Activity</td>
<td></td>
<td>Product</td>
<td></td>
<td></td>
<td>Lifecycle</td>
<td></td>
<td></td>
<td>Direction</td>
</tr>
<tr>
<td>ISO/IEC 1074-1997</td>
<td>Activity</td>
<td>Product</td>
<td></td>
<td>Activity group</td>
<td>Lifecycle process</td>
<td></td>
<td></td>
<td>Phase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMBOK</td>
<td>Staff</td>
<td>Task</td>
<td></td>
<td>Deliverable</td>
<td>Activity</td>
<td></td>
<td>Lifecycle</td>
<td></td>
<td></td>
<td>Technique</td>
</tr>
</tbody>
</table>

SPEM – ad/2001-06-05  75/78  rev 2.1c  6/6/2001
Annex 2: Example from the DMR Macroscope

Following is a Software Process Engineering Model instantiation example. This example only represents a portion of a typical information system delivery process. Process metamodel (M2) classes, associations and attributes are represented in courier while the corresponding M1 instances appear in bold times font.

Phase: Preliminary Analysis
Process: Information System Delivery Process

Subactivities
Iteration: First Joint Requirements Planning (JRP) Workshop
Subactivities
Activity: Define Owner Requirements
ProcessRole: System Architect
ActivityParameters [kind: input]
   WorkProduct: EnterpriseArchitecture
ActivityParameters [kind: output]
   WorkProduct: Assessment of Current System [state: initial draft]
   WorkProduct: Owner Requirements [state: initial draft]
Steps
   Step: Define objectives based on stated needs
   Step: Define the key issues
   Step: Determine the relevant enterprise principles

Activity: Draft Owner Models
ProcessRole: System Architect
ActivityParameters [kind: input]
   WorkProduct: Assessment of Current System [state: initial draft]
   WorkProduct: Owner Requirements [state: initial draft]
ActivityParameters [kind: output]
   WorkProduct: Business Structure [state: initial draft]
   WorkProduct: Business Dynamics [state: initial draft]
Steps
   Step: Determine System context
   Step: Model structural and dynamic aspects of the enterprise
   Step: Define work resources
   Step: Explore with prototypes

Activity: Define User Requirements
ProcessRole: System Architect
ActivityParameters [kind: input]
   WorkProduct: Assessment of Current System [state: initial draft]
   WorkProduct: Owner Requirements [state: initial draft]
ActivityParameters [kind: output]
   WorkProduct: User Alternatives [state: initial draft]
   WorkProduct: User Principles [state: initial draft]
Steps
  Step: Consider user interface aspects
  Step: Consider distribution aspects
  Step: Explore with prototypes

Activity: Draft User Models
  ProcessRole: System Architect
  ActivityParameters [kind: input]
    WorkProduct: User Alternatives [state: initial draft ]
    WorkProduct: User Principles [state: initial draft ]
    WorkProduct: Business Structure [state: initial draft ]
    WorkProduct: Business Dynamics [state: initial draft ]
  ActivityParameters [kind: output]
    WorkProduct: System Structure [state: initial draft ]
    WorkProduct: System Dynamics [state: initial draft ]
Steps
  Step: Determine System context
  Step: Model structural and dynamic aspects of the system
  Step: Define work resources
  Step: Explore with prototypes

Activity: Define Developer Requirements
  ProcessRole: Technical Architect
  ActivityParameters [kind: input]
    WorkProduct: User Alternatives [state: initial draft ]
    WorkProduct: User Principles [state: initial draft ]
  ActivityParameters [kind: output]
    WorkProduct: Developer Alternatives [state: initial draft ]
    WorkProduct: Developer Principles [state: initial draft ]
    WorkProduct: Technology Infrastructure
      [state: initial draft ]
Steps
  Step: Revise work process and class definitions
  Step: Revise user interface models

Activity: Draft Developer Models
  ProcessRole: Technical Architect
  ActivityParameters [kind: input]
    WorkProduct: Developer Alternatives [state: initial draft ]
    WorkProduct: Developer Principles [state: initial draft ]
    WorkProduct: Technology Infrastructure
      [state: initial draft ]
    WorkProduct: System Structure [state: initial draft ]
    WorkProduct: System Dynamics [state: initial draft ]
  ActivityParameters [kind: output]
    WorkProduct: Software Architecture [state: initial draft ]
    WorkProduct: Persistent Information [state: initial draft ]
Steps
  Step: Define process and data aspects of the system
  Step: Consider user interface aspects
  Step: Consider distribution aspects
  Step: Explore with prototypes
Subactivities
  Iteration: Second Joint Requirements Planning (JRP) Workshop
  Subactivities
    Similar to First Joint Requirements Planning (JRP) Workshop iteration:
    - reuse and cumulate existing WorkProduct assets as input to activities
    - change «initial draft » output WorkProduct states with «revised draft »

Phase: System Architecture
  Process: Information System Delivery Process
  Subactivities
  Iteration: First Joint Application Design (JAD) Workshop
  Subactivities
    Activity: Revise User Models
      ProcessRole: System Architect
      ActivityParameters {kind : input}
        WorkProduct : System Structure [state: revised draft ]
        WorkProduct : System Dynamics [state: revised draft ]
      ActivityParameters {kind : output}
        WorkProduct : System Structure [state: revised ]
        WorkProduct : System Dynamics [state: revised ]
    Steps
      Step : Revise work process and class definitions
      Step : Revise user interface models
      Step : Realize/improve prototype

etc.

Phase: System Architecture
  Process: Information System Delivery Process
  Subactivities
  Iteration: Second Joint Application Design (JAD) Workshop
  etc.

•