MOMOCS

Model driven Modernisation of Complex Systems

DELIVERABLE #D32
METHODOLOGY STANDARDS

Dissemination Level:	Public
Work package:	WP3
Lead Participant:	SOFT
Contractual Delivery Date:	M10
Document status:	Final
Preparation Date:	30th June, 2007 Month

INDEX

1 SCOPE 4

1.1 PURPOSE OF THE DOCUMENT 4
1.2 DOCUMENT STRUCTURE 4

2 OVERVIEW 5

2.1 BACKGROUND CONCEPTS 5
2.1.1 Software life cycle model 5
2.1.2 Software process model 6
2.2 A MODELLING APPROACH 7
2.2.1 UML Profiles 10

3 SPEM 12

3.1 DESCRIPTION 13
3.2 TERMINOLOGY 16
3.3 SPEM AND OTHER APPROACHES 20

4 CONCLUSION 22

5 ANNEXES 23

5.1 REFERENCE DOCUMENTS 23

INDEX OF FIGURES

Figure 1: Layers of modelling [SPEM] 9
Figure 2: Example of UML Profile [FaM04] 11
Figure 3: SPEM conceptual model 13
Figure 4: SPEM metamodel 15
Figure 5: ProCube activity icon 16
Figure 6: ProCube role icon 17
Figure 7: ProCube phase icon 17
Figure 8: ProCube step icon 18
Figure 9: ProCube workdefinition icon 18
Figure 10: ProCube workproduct icon 19

1 Scope

1.1 Purpose of the Document

The purpose of this document is to introduce the main background concepts of *software process engineering* and to provide a detailed description of the OMG standard used to formalize the Xirup methodology: the Software Process Engineering Metamodel (SPEM). In addition, basics of meta-modelling and UML profiling are given in order to help in understanding the approach for XIRUP meta-model specification.

The document is strongly linked with D31 where the Xirup process is formally described through SPEM diagrams. The meta-model presented in D41 is specified with MOF and it is also presented in this document. UML2 Profiles are described in order to illustrate the metamodel implementation with UML2 Profiles.

1.2 Document Structure

This document is organised as follows.

Section 2 presents a brief explanation of the "software life cycle" and the software process" models. It also describes the Meta-model Facility (MOF) frameworks and UML profiles, which are used to specify SPEM.

Section 3 describes SPEM providing rationale behind its choice in MOMOCS project.

Section 4 highlights the conclusions reached.

Section 5 summarises the bibliography consulted for writing the present deliverable.

2 Overview

This section presents a brief explanation of the "software life cycle" and the "software process" models for what regards software process engineering. Finally, it offers a short description of Meta-model Facility (MOF) framework and UML profiles, which are used to specify SPEM.

2.1 Background concepts

Before considering some process engineering concepts we should first define what a "software process" is: [CERN] states that "a software process is the set of actions, tasks and procedures involved in producing a software system, throughout its life cycle". Modelling the software process facilitates its management with the support of process management tools.

2.1.1 Software life cycle model

A *software life cycle model* is either a descriptive or prescriptive characterization of how software is or should be developed [SCA01].

A **descriptive model** describes the history of how a particular software system was developed.
A **prescriptive model** prescribes how a new software system should be developed.

Prescriptive models are used as guidelines or frameworks to organize and structure how software development activities should be performed, and in what order.

Typically, it is easier and more common to articulate a prescriptive life cycle model for how software systems should be developed. This is possible since most such models are intuitive or well reasoned. This means that many details that describe how a software system is built in practice can be ignored, generalized, or deferred for later consideration. This, of course, should raise concern for the relative validity and robustness of such life cycle models when developing different kinds of application systems, in different kinds of development settings, using different programming languages, with differentially skilled staff, etc. However, prescriptive models are also used to package the development tasks and techniques for using a given set of software engineering tools or environment during a development project.

Descriptive life cycle models, on the other hand, characterize how particular software systems are actually developed in specific settings. As such, they are less common and more difficult to articulate for an obvious reason: one must observe or collect data throughout the life cycle of a software system, a period of elapsed time often measured in years [SCA01].

2.1.2 Software process model

In contrast to software life cycle models, *software process models* often represent a networked sequence of activities, objects, transformations, and events that embody strategies for accomplishing software evolution. Such models can be used to develop more precise and formalized descriptions of software life cycle activities. Their power emerges from their utilization of a sufficiently rich notation, syntax, or semantics, often suitable for computational processing [SCA01].

The software process is a process with special features, which are derived from the particular complexity of the resulting software products. In a company or in an

application domain, processes in different projects usually follow certain common patterns, because good practices are formally recognised, or because of the existence of some standards. The representation of a software process model allows to describe these common features and promotes homogeneity.

The software process is based on a lifecycle model, and involves all necessary elements (technologies, personnel, artefacts, procedures, etc.) that are related with the activities in a software product life.

Note that the software process has to consider two interrelated processes: *Production process* (development and maintenance of software) and *Management process* (planning and control of resources that are required for the production process).

Modeling the software process facilitates its understanding and communication, support for its management, quality assurance, and the availability of support tools.

Recently, it becomes common to accept the coexistence of different software process models or the need to customize generic software process models for the development of applications in concrete domains. This has largely motivated the definition of process modeling languages such as SPEM by OMG or SEMDM (ISO/IEC 24744 standard). These standards should facilitate the availability of tools to specify and manage software processes.

2.2 A modelling approach

UML (Unified Modeling Language) represents the killer-language that permitted to move from code-oriented to model-oriented software production techniques.

The need to specify more formally UML gave rise in OMG to the definition of languages that allow the specification of modeling languages. This need has been reinforced by the recognition that UML is not the only possible modeling language in the software development landscape. The result has been the MOF (Meta Object

Facility) [BEZ04]. To avoid a variety of different non-compatible meta-models being defined and independently evolving (data warehouse, workflow, software process, etc.), there was an urgent need for a global integration framework for all meta-models in the software development area. To this end, a language for defining meta-models was provided, i.e. a meta-meta-model (layer M3 in Figure 1) while each meta-model defines a language for describing a specific domain of interest (layer M2 in Figure 1).

For example, UML addresses the need to model the artifacts of object-oriented software systems. Some other meta-models may address domains like legacy systems, data warehouses, software process, organization, tests, quality of service, party management, etc. Each is important and their numbers keep growing. They are defined as separate components and many relationships exist between them [BEZ04]. MOF defines a four layer metadata architecture, which can be applied to SPEM as depicted in **Error! Reference source not found.**. The lower layer (M0) represents a concrete project, which is an instance of a concrete process model (in M1 layer). The process model is specified using a language, such as SPEM. The specification of this language is in M2 layer. And it is specified using a language of M3 layer (MOF in this case).

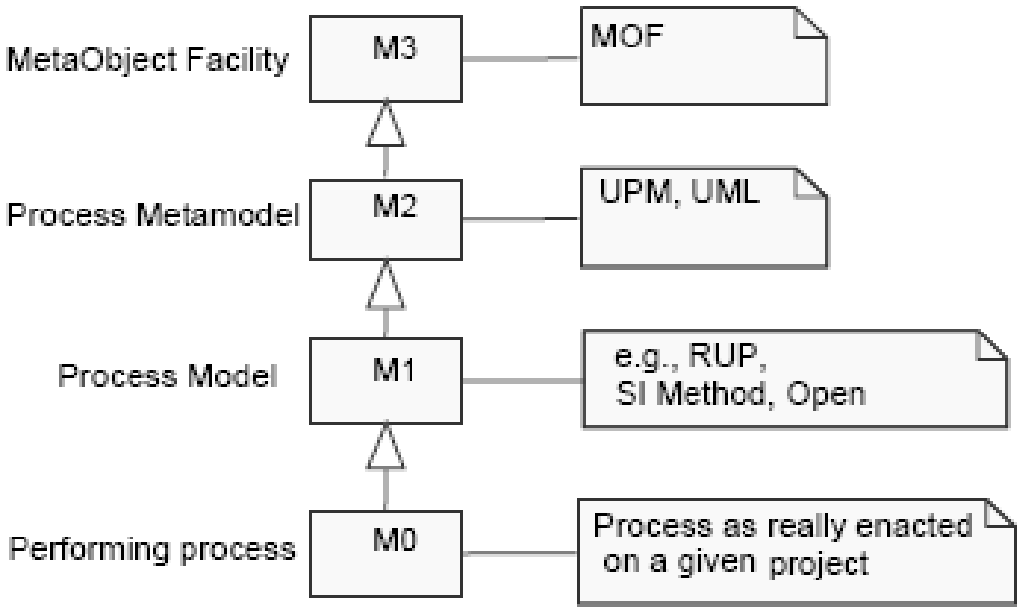

Figure 1: Layers of modelling [SPEM]

More concretely, the SPEM specification is structured as a UML profile (see next section) and provides a complete MOF-based meta-model for software process engineering. Thus, it permits to formally describe a full-fledge software development methodology and consequently, a complete software modernization set of methods as required by Xirup.

Specified by UML2 Infrastructure, MOF provides a comprehensive approach for defining new modelling languages. In the context of XIRUP, MOF is used for definition of XIRUP metamodel, which is presented in [D41].

2.2.1 UML Profiles

As presented above, MOF is common way for definition of new modelling concepts. However, for implementing this concept a solid methodological and modelling tool support is required. The UML Profiles presented in this section provides an approach that help to implement new concepts on the already established, well-developed standard basis. In our approach, the XIRUP metamodel represents the language concepts, while UML Profiles are used for implementing them on the UML2 basis.

UML Profiles is the extension mechanism for UML modelling. This mechanism is also a basis for customisation Modelling Tool, like Objecteering by Softeam. The UML Profiles allows implementing new modelling concepts and functionalities based on the standard modelling functionalities provided by tools. UML elements and diagrams can be specialized for implementing new modelling languages. This customisation can be done to reduce complexity of UML or reduce the scope of the modelling for a concrete usage. For example, for the requirements definition UML foresees Use-Case diagrams and Sequence Diagrams. The users that specify requirement may be provided with a restricted GUI for working with these concrete diagrams only. For users specifying conceptual models a dedicated interface and restrictive language can be specified for helping to implement a specific methodology.

UML can easily be customized by using a set of extension mechanisms that UML itself provides in order to build new metamodel. More precisely, the Profiles package included in UML 2.0 defines a set of UML artifacts that allows the specification of an MOF model to deal with the specific concepts and notation required in particular application domains (e.g., real-time, business process modelling, finance, etc.) or implementation technologies (such as .NET, J2EE, or CORBA). The profiling mechanism allows to customize any MOF-like metamodel and not only UML-

defined ones. On the same manner, a UML Profile can also define another UML Profile.

UML Profiles are defined in terms of three basic mechanisms: *stereotypes*, *constraints*, and *tagged values* [FaM04].

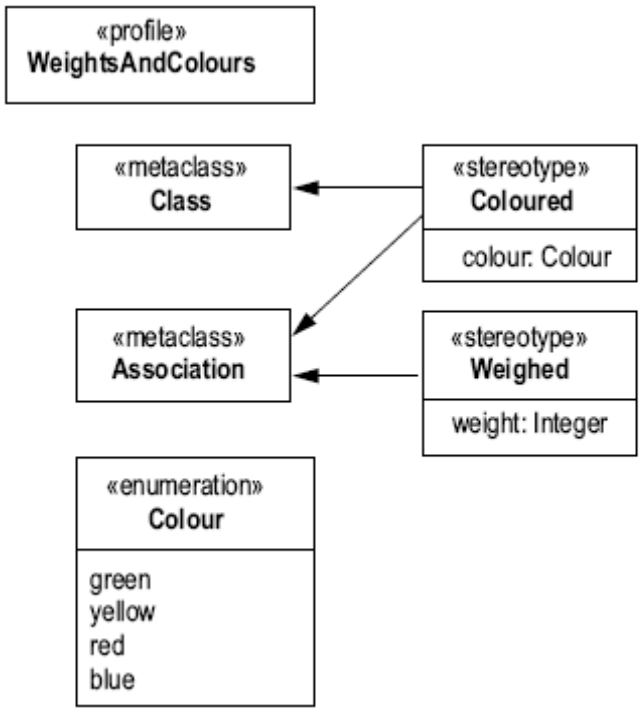

Figure 2: Example of UML Profile [FaM04]

3 SPEM

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 [SPEM]. However, some tools can take as input processes that have been specified with SPEM, and manage them (for instance, ProCube).

A guiding principle for SPEM has been simplicity in the following sense: *"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" [SPEM].*

3.1 Description

SPEM considers a software development process as collaboration between abstract active entities called *process roles*, which perform operations called *activities* on concrete, tangible entities called *work products* [SPEM]. This is represented in the UML diagram in 4.

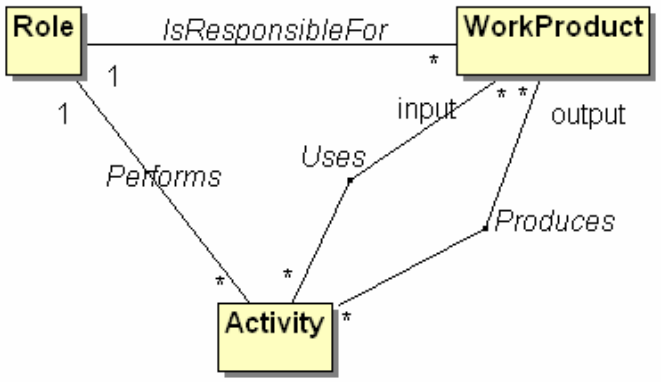

Figure 3: SPEM conceptual model

These three basic elements are refined to provide the SPEM metamodel, which is shown in the diagram in **Error! Reference source not found.**4.

The core idea is that different roles act upon one another or collaborate by exchanging products and triggering the execution for certain activities.

The global goal of the software process is to lead a set of products and artefacts to a well-defined state [CCCC06].

A *WorkDefinition* describes the execution, the operations performed, and the transformations enacted on some *Work Products* by the *roles*. It can also be decomposed reflexively, generating a set of subworks. The *WorkDefinition* has four

specializations, respectively *Activity*, *Iteration*, *Phase* and *LifeCycle*: an Activity can be divided into *Steps* while a LifeCycle is a sequence of Phases and Iterations.

A *ProcessPerformer* is the role that is the unique responsible for a certain WorkDefinition; on the other side, a set of *ProcessRoles* has the possibility to assist the main role to perform an Activity.

SPEM also defines two major concepts, *Process* and *Discipline*. A Process corresponds to the root of a process model from which a tool can do the transitive closing of a complete process. A Discipline allows, within the process, to partition activities according to a common "theme". The output products of each of the activities of a discipline must be categorized under this same theme [CCCC06].

Diagram 5 shows the main elements that facilitate the description of a process using SPEM; the highlighted meta-objects relates to the elements just described in the above paragraphs.

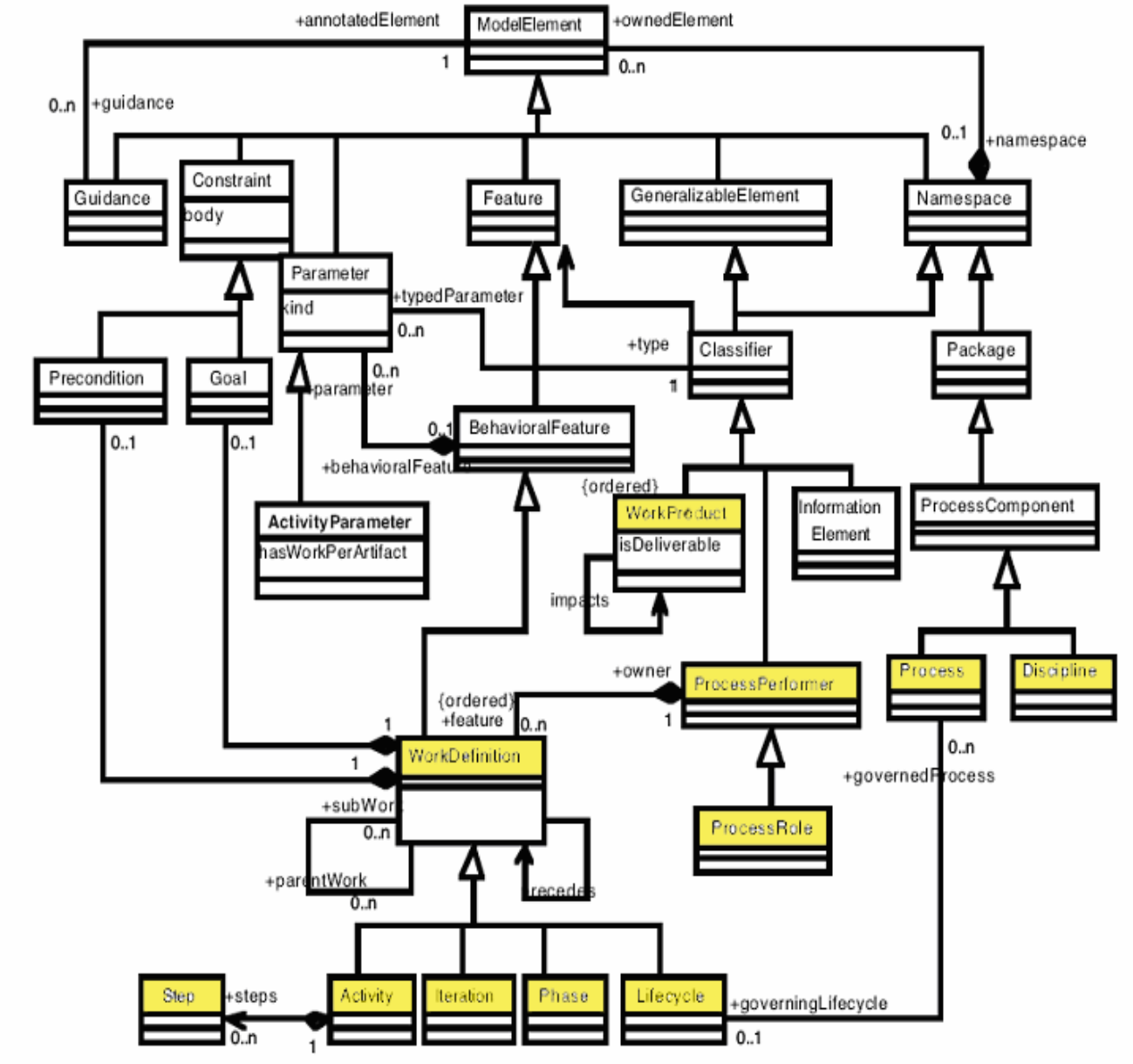

Figure 4: SPEM metamodel

3.2 Terminology

Taking into account the [SPEM], a complete description of metamodel elements is given in the following bullets. In some cases, such elements are presented along with their corresponding ProCube graphical representation in order to enforce the link with D31 where Xirup methodology is introduced just with ProCube diagrams.

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

Figure 5: ProCube activity icon

- **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.

- **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.

Figure 6: ProCube role icon

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

Figure 7: ProCube phase icon

- **Process:** A Process is a complete description of a software engineering process, in term of Process Performers, Process Roles, Work Definitions, Work Products, and associated Guidance.

- **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.

Figure 8: ProCube step icon

- **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.

Figure 9: ProCube workdefinition icon

- **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.

Figure 10: ProCube workproduct icon

3.3 SPEM and other approaches

There are other proposals for the specification of development processes.

ISO has defined the Software Engineering Metamodel for Development Methodologies (SEMDM) [ISO 24744]. The purpose of this standard is to provide a comprehensive meta-model for the specification of all the concepts necessary for modelling and creating methodologies. It can be used to generate and underpin a unique methodology or used in a method engineering context to create endeavour-specific methodologies. This meta-model relies on the use of powertypes [GonHen06] as its underpinning conceptual architecture, which is different from the OMG instantiation-linked multi-layer architecture (usually denoted as M0, M1, M2 and M3 levels). Three layers are used in ISO/IEC 24744: endeavour (where people work), method (where practices are determined) and meta-model (where practices are formally defined).

The use of powertypes is claimed to solve some inconsistencies that may appear when using MOF. Powertype metamodelling intends to consider both the perspective of software developers (users of methodologies) and method engineers (users of metamodels and creators of methodologies), in an integrated way. This is done by allowing to define pairs of classes to model method-level concepts (for instance, TaskKind) and project-level concepts (such as Task). Each of these pairs composes a powertype pattern that ties together both concepts and, at the same time, allows for independent usage. [HenGon05].

Although from a theoretical point of view the new ISO standard based on powertypes could be superior, there are no tools supporting it. This is an important reason to consider the use of SPEM instead, as it is widely available. In concrete, in the Momocs project one of the partners is distributing a tool, called ProCube, which allows to specify processes and enact them.

Another interesting framework is the Eclipse Process Framework (EPF). EPF aims at producing a customizable software process engineering framework, with exemplary process content and tools, supporting a broad variety of project types and development styles. EPF has several advantages with respect to similar tools such as a wider acceptance, because it is open software supported by Eclipse Foundation, its greater functionality, and better adaptability. In fact, there are plans to adopt SPEM 2.0 once this is released.

The associated tool, EPF Composer, provides an environment for defining, tailoring, managing, and communicating development processes. It facilitates the creation of a process by defining who does what, when, and how. It can also provide guidance, templates, and other supporting material and publish the information as a website on a corporate intranet.

By separating the *method content*, which defines the roles, work products, tasks, and associated guidance from the *process*, which defines the phases, iterations, activities, and tasks in the context of a work breakdown structure, one can reuse content across projects, phases, and iterations, tailoring the life cycle to meet the specific needs of the project team. This approach effectively supports top-down process definition (phases, iterations, and activities), bottom-up process definition (roles, work products, and tasks), or a combination of the two.

4 Conclusion

The SPEM metamodel has several strong points and a few week points that make it suitable to describe XIRUP and the MOMOCS approach.

SPEM weak points are related to its complexity and the fact that semantics is only partially formalised. For example the *ProcessPerformer* [BEN05] is not formally defined. SPEM also lack enactment and planning support. All these limitations do not seem serious enough to advise against using SPEM in MOMOCS.

On the other hand there are several characteristics of SPEM that make it a good fit for the description of XIRUP. SPEM is compliant to MOF and can be extended with UML2.0 elements for detailed process definition. It enjoys wide support from OMG and is the elective choice by industry (ORACLE, MacroMedia and others) when it is necessary to model complex processes.

SPEM has thus been chosen for describing XIRUP in D31.

5 Annexes

5.1 Reference Documents

[SCA01] "Process Models in Software Engineering",Walt Scacchi, Institute for Software Research, University of California, Irvine-February 2001

[BER04] "Applying The Basic Principles of Model Engineering to The Field of Process Engineering" Jean Bézivin and Erwan Breton, October 2004

[SPEM] "Software Process Engineering Metamodel Specification" Version 1.1 formal/05-01-06, January 2005
http://www.omg.org/docs/formal/05-01-06.pdf

[ISO 24744] "Software Engineering Metamodel for Development Methodologies (SEMDM)", ISO/IEC 24744

[CERN] http://ipt.web.cern.ch/IPT/Papers/Sopron94/CSCproceedings_11.html

[OST87] "Software processes are software too", ICSE'87, L. Osterweil Monterey, Ca

[BEZ04] "In Search of a Basic Principle for Model Driven Engineering" Jean Bèzivin, October 2004

[FaM04] "An Introduction to UML Profiles", Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno, October 2004

[CCCC06] "Towards a rigorous process modeling with SPEM", Benoit Combemale, Xavier Crégut, Alain Caplain, Bernard Coulette, ICEIS April 25, 2006 www.combemale.net/research/phd/2006/iceis250406-CCCC-poster401.pdf

[BEN05] Bendraou, R., Gervais, M.-P., and Blanc, X. (2005). Uml4spm : A uml2.0-based metamodel for software process modelling. In MoDELS'05, volume 3713, pages 17–38. Springer-Verlag.

[GonHen06] Cesar Gonzalez-Perez, Brian Henderson-Sellers: A powertype-based metamodelling framework. Software and System Modeling 5(1): 72-90 (2006)

[HenGon05] Brian Henderson-Sellers, Cesar Gonzalez-Perez: The Rationale of Powertype Powertype-based Metamodelling to Underpin nderpin Software D Development evelopment Methodologies. APCCM 2005: 7-16