Observations on Modeling Software Processes with SPEM Process Components

Antero Järvi and Tuomas Mäkilä

University of Turku,
Department of Information Technology,
FI-20014 University of Turku, Finland

Abstract. OMG's standard for software process modeling (SPEM) contains an element, ProcessComponent, that could be used as a reusable element to assemble end-to-end software processes. However the composition mechanism and the nature of process components is not well defined in SPEM. In addition the organization of process components is not straightforward. In this paper we describe an experiment of using CMMI Process Areas and Rational Unified Process disciplines as a basis for structuring process components. The two approaches produced process components that could not be used together. We conclude that in order to achieve reusable process components, the components must be defined using a common process framework. Further, we give propositions for organizing process components that facilitate component reuse and composition.

1 Introduction

All organizations involved in developing software need to organize, manage and support the development work. Organization's *software development process* ties together all activities and practices addressing this need. Recent years have shown a clear trend of growing emphasis on the software process – the process is thought to be a key factor in ensuring high product quality, achieving accurate time and cost estimates, providing cost efficiency in software development and coordinating large development efforts. In other words, software process has and increasingly important role in achieving and maintaining competitiveness in software business.

Software processes can be defined in many levels of detail, ranging from processes defined implicitly in project management, tools and work practices up to high-ceremony explicitly defined and documented processes. Explicit process description opens the route for *software process engineering* (SPE), consisting of modeling, authoring, tailoring and enacting processes. A natural part of SPE is *software process improvement* (SPI), a deliberate effort to document and modify organizations software processes to increase its competitive strength. However, manual process authoring and tailoring may be impractical. SPI that involves constant change of software processes becomes prohibitively expensive. The overhead cost of SPE and SPI can be brought down by tool support and process automation, both enabled by a process modeling language.

Recently, OMG published a standard for software process modeling, *Software Process Engineering Metamodel* (SPEM) [1], that provides the required formalism for process engineering tools. This paper investigates the suitability and use of SPEM for modeling reusable process components. Section 2 briefly introduces SPEM and related concepts of organizations process engineering. In section 3 we describe the modeling experiment that was carried out. We focus on two frameworks: Capability Maturity Model Integration (CMMI) [2] and Rational Unified Process (RUP) [3]. CMMI is a maturity model, which provides a roadmap for practicing SPI in an organization, RUP is a *de facto* process standard, which defines a software process and its roles, activities, and work products at detailed level. In section 4 the problems of SPEM in this context are summarized and a proposition of process component organization to alleviate the problems is made. This paper is based on work done in the ReProCo-project[1] aiming at constructing reusable process components for various software projects.

2 Background

The background of software process modeling and related technologies are comprehensively reviewed in [4]. Since then the research has somewhat focused around SPEM.

2.1 Software Process Engineering Metamodel (SPEM) overview

The SPEM specification is used for describing a concrete software development process or a family of related software development processes [1]. SPEM is structured as an UML profile and provides a complete OMG Meta-Object Facility (MOF) metamodel. SPEM also defines a notation for its concepts, defined as UML stereotype icons. The main idea of SPEM based process representation is the interplay of three basic elements: *ProcessRoles* that are responsible for and execute *Activities* that consume and produce *WorkProducts*. ProcessRoles, WorkProducts and Activities are all *process definition elements*. Relationships between these basic elements are illustarted in Figure 1.

SPEM defines *LifeCycle*, *Phase* and *Iteration* that are used for dynamic structuring of the process. A LifeCycle defines the ordering of Phases, which in turn can contain Iterations. A Process must have one LifeCycle.

SPEM also defines elements that are meant for organizing other process elements from the viewpoint of process authoring, assembly and reuse. *Packages* are concerned with dividing one or more process descriptions into self-containing parts. These parts can then be placed under configuration and version management and used in assembling and tailoring software development processes. *ProcessComponents* are specializations of Packages. A ProcessComponent is an internally consistent and self-contained chunk of process description that may be

[1] Part of E!3320 project, in co-operation with Genestia Group Inc. - Neoxen Systems and Devera Software Development Center.

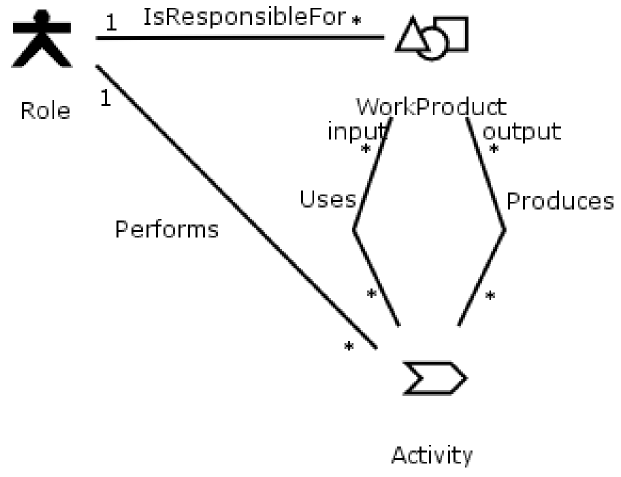

Fig. 1. The core elements of SPEM and their relationships. [1]

reused with other ProcessComponents to assemble a complete process. ProcessComponents can import a non-arbitrary set of process definition elements. The *Discipline* is a specialization of ProcessComponent and is used for representing activities in a common area (corresponding to e.g. Core Workflows in the Unified Process). *Process* finally is a also a specialization of ProcessComponent that is intended to stand alone as a complete end-to-end process.

The assembly of processes is done by composition of ProcessComponents. This requires *unification* of the ProcessComponents. Corresponding output and input WorkProducts must be unified, as well as ProcessRoles and possibly other elements that are used in more than one ProcessComponent. The details of unification are not defined in SPEM.

2.2 OPA, OSSP and software process frameworks

Capability Maturity Model Integration (CMMI) [2] introduces concepts of *Organization's Process Assets* (OPA) library and *Organization's Set of Standard Processes* (OSSP). OPA library is a loosely tied collection of "process assets that are potentially useful to those who are defining, implementing, and managing processes in the organization". OSSP defines a set of processes that "guide all activities in an organization" and "cover the fundamental process elements". OSSP forms a base for organization's project-level tailoring whereas more detailed definitions for process elements can be found from the OPA library. Both OPA Library and OSSP are highly organization specific, depending on the business goals and characteristics of the organization. Similar tailoring mechanisms are also suggested by other major process standards (ISO 12207 [5], RUP [3]).

OSSP is not normally developed from scratch nor simply put together from existing and past projects – instead, so called software process frameworks should be used as guidelines. There are numerous process frameworks available, each focusing on certain types of software development organizations. A partial overview of the software process frameworks and their categorization can be found from the Frameworks Quagmire [6] [7].

2.3 Levels of process modeling

The process modeling language must support modeling at the framework level, at the OSSP level, and at the project level. In Figure 2 these different levels of process modeling are shown with respect to the four-layered organization of SPEM [1]. Layer M2 defines the SPEM language as OMG MOF model. Layer M1 contains process models created with SPEM language, and layer M0 is the enacted software process. Different levels of process modeling are not directly supported by the layering, specifically layer M1 must be divided into two sub-layers: 1) general process modeling and description that models elements of process frameworks, and 2) OSSP that structures modeled process elements so that reusability of these assets is achieved. In 'Project-level tailoring', level M0 performing processes are enacted by selection, composition and tailoring of reusable process components of OSSP M1 sub-layer. These components in turn contain modeled elements from ProcessFramework M1 sub-layer and are tailored to meet organizations need for different types of processes.

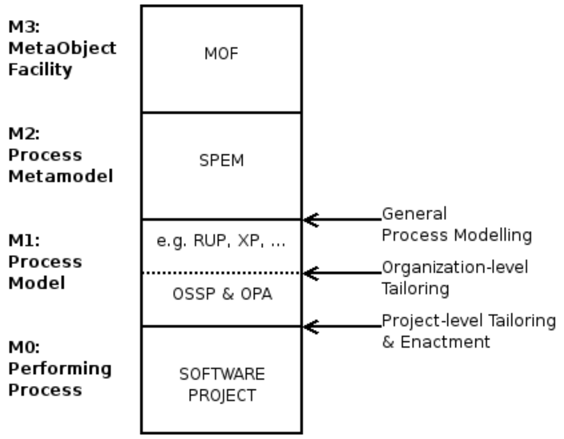

Fig. 2. Process modeling levels contrasted with the OMG SPEM language organization as an UML profile.

Support for this kind of sub-layer organization of organizations process assets is not well included in SPEM standard. The basic SPEM construct for structuring process elements, ProcessComponent, is simply a self-contained Package. SPEM does not define how the underlying process or process framework should be decomposed into components. However, this decomposition largely determines how reusable and how pluggable the process components become. Most process frameworks define a natural decomposition, e.g. Process Areas in CMMI or Disciplines in Unified Process. SPEM standard mentions e.g. Disciplines of Rational Unified Process as possible candidates for ProcessComponents. It appears, however, that this proposed framework decomposition does not yield very reusable process components. The whole issue of framework decomposition to define conceptual organization of the OSSP is far more complicated than we and apparently the SPEM standard anticipated. We will get back to this issue in Section 4.

3 The modeling experiment

3.1 Modeling CMMI Process Areas as process components

The original plan was to model CMMI *Requirement Management* (REQM) and *Requirements Development* (RD) Process Areas [2] as two process components. CMMI was chosen because it structures software process in widely adopted and accepted way. The reasoning was that with this kind of approach, reusable process component 'stubs' would form the OSSP. The conceptual organization of the OSSP would follow CMMI Process Areas, thus CMMI compliance of the tailored process would be straightforward to show. The contents for these 'stubs', the OPA of the organization, would be obtained from other software process frameworks and the components would be mutually compatible as long as they comply to corresponding requirements of CMMI. This would have enabled assembling a software process from several different process frameworks and using most suitable parts of each.

The visible interface of a process component in SPEM is defined by components inputs and outputs. Thus, in order to model Process Areas as process components, we analyzed what process elements were consumed or produced by a particular Process Area, i.e. its inputs and outputs. Modeling was done at the CMMI Specific Practice level. Specific Practices are brief statements on what is required from a software process in order be compliant with regard of a particular Process Area. Special Practices essentially form the backbone of each Process Area. It became apparent that Specific Practices, despite their activity-like nature, describe only the minimum set of Process Areas requirements. In order to construct a process component that could be used in a real software process, Process Areas requirements must be complemented with a software process framework that defines the missing detailed content of the process model, in particular the exact set of inputs and outputs of the Process Area. This suggests that process components can not be modeled based solely on the CMMI Specific Practices. The textual guidelines and work product recommendations of CMMI provide some of the missing content of the process component, but fall short on providing detailed interface descriptions. The interface detail must be obtained from another software process framework. This hints that CMMI Process Area based components will not have high reusability and can not be used to form an OSSP.

3.2 Integrating RUP and CMMI process components

To set the underlying software process methodology and provide detailed content for CMMI Process Area based process components, we chose commercial *Rational Unified Process* framework (RUP) [3]. The reason for this choice was that the framework is widely adopted, there is enough reference material about the framework available and it is based on the more academic Unified Process framework [8]. Also RUP involves many of the generally accepted properties of modern software process: iterative, incremental, use-case driven and architecture centric to mention a few [8].

We began the modeling experiment by mapping relevant parts of RUP methodology into CMMI Process Area based process components. Because RUP and CMMI are structured differently and CMMI generally speaking operates at a higher level of abstraction, process element level correspondences between these two frameworks had to be established. This required interpretation of the elements roles in the two frameworks, since directly corresponding elements could not be always found. In CMMI, Specific Practices were chosen as elements that were mapped into RUP activities and work products. Figure 3 illustrates a mapping of CMMI REQM Process Area Specific Practices into RUP. The mapping shows that the practices of the CMMI REQM Process Area do not map into a single RUP discipline, even though only the Requirements discipline should contain requirements related activities in RUP. It should be noted that only Special Practices 1.1. and 1.2. are shown in Figure 3 The remaining three Special Practices of REQM are similarly scattered throughout the RUP disciplines. Inverse mapping from the RUP Requirements discipline to REQM and RD Process Areas strengthened this finding: only part of the practices in RUP Requirements discipline could be mapped back into CMMI REQM or RD.

4 Findings

In our experiment we found two different impediments of full scale use of SPEM process components as a basis for an OSSP: 1) shortcomings of SPEM standard regarding process components and 2) lack of established practices of organizing process components at conceptual level.

4.1 SPEM process component shortcomings

SPEM language is well-designed to model basic process definition elements. However the definition of ProcessComponent is ambiguous: requirement of self-contained components means that there must be no 'RefersTo' dependencies

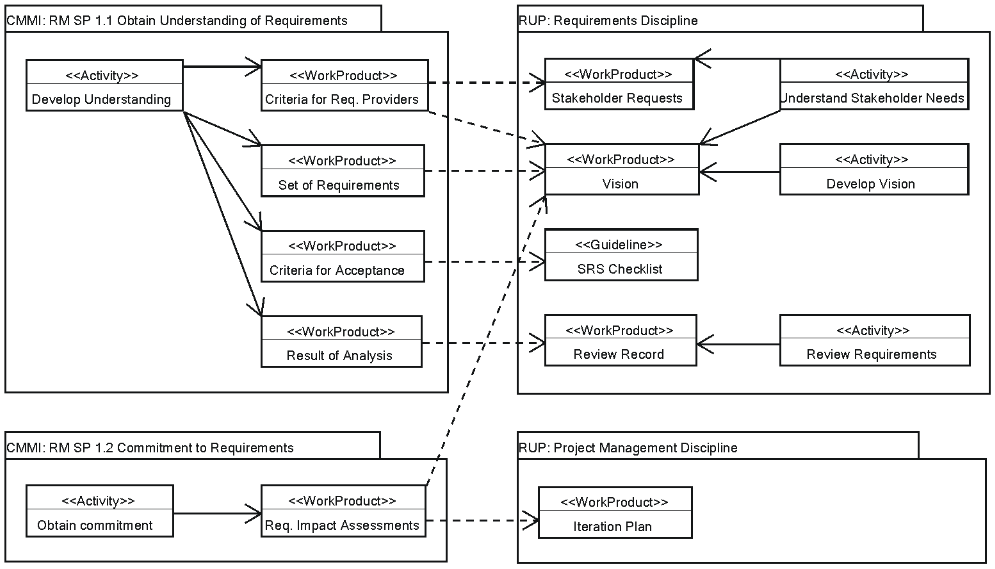

Fig. 3. An exemplar mapping of CMMI practices to RUP activities illustrate how these two frameworks structure the requirements management differently.

from within the component to elements not within the component. Other dependency types, e.g. 'Import' are allowed. The semantics of 'RefersTo' and 'Import' can be interpreted in many ways. Figure 4 shows one possible use of the dependencies. [1] The absence of any examples in the standard illustrating the intended use of these dependencies will lead to different ways of composing process models from process components in different organizations.

Composition of process components is done by *unification*. SPEM states that at least output work products of component P1 and input work products of component P2 must be made identical in order to combine P1 and P2. In addition, according to SPEM other elements may possibly be also unified, such as Process-Roles, Templates, and so on. SPEM suggests that composition of components could only be fully automated if they originate from a common family, so that unification could be automated. Otherwise the unification would involve human intervention consisting re-writing of the elements [1]. However, SPEM does not provide any explicit support for component composition and unification – the issue is left open.

From the viewpoint of process component reusability, the assembly mechanism may allow too much variation and jeopardize component portability between two OSSPs. Also tool independency is compromised as tool vendors may interpret SPEM standard composition mechanisms differently.

4.2 Conceptual organization of process components

Process components could be formed from process frameworks by using their existing organization to identify components. As an example, making process components out of RUP disciplines is certainly a natural decomposition. However, looking closer at how one discipline in RUP interacts with other disciplines in RUP, we clearly see that the composition interface of the component will become very complex. The work carried out in a discipline, expressed as an activity flow, is connected to many other disciplines in all four phases of RUP. Further, phases contain iterations, and many disciplines are active during one iteration. Roles, tools, guidances and templates are shared between disciplines, and continuous, integral workflows of a single worker continue seamlessly from one discipline to another. Expressing such a complicated and versatile relationship between process components, with the simple support that SPEM offers, is challenging.

It is questionable whether these kind of large process components are useful at all in an OSSP. Achieving reusability in general requires that the reused component has a well defined task or responsibility, and has a simple interface - otherwise it will not be pluggable without considerable manual tailoring. It may very well be the case that these problems originate from the foundational level, not specifically from SPEM. Process components that are formed from RUP Disciplines or CMMI Process Areas appear not to be reusable or even usable. The only situation where the composition is gets manageable is a simple waterfall type sequencing where process components represent the phases of the life-cycle and are connected via major mile-stone work products. Note that in this case disciplines (e.g. requirements, design, testing) in fact coincide with the phases.

The other possible approach to create process components is to begin by defining process elements of a process framework: work products, roles, guidances etc., and model the dependencies between these basic elements. These elements have high reusability and process independence, and can thus be used to form the OPA. SPEM Package is a natural choice to model this library of stable process elements. The OSSP has the role of modeling the dynamic part of the process. It consists of rather small process components, packaging together a cohesive flow of activities, the participating roles, guidances and tool mentors, templates, and related work products. Most of this content could be taken from the OPA using 'Import' dependency. Elements that remain internal to only one process component need not be added to OPA. The main reason for this arrangement is the unification of process elements shared by several process components.

'RefersTo' dependency is used in the OPA to indicate which elements must be imported together due to dependencies. The situation is illustrated in Figure 4. For instance, importing 'Use Case Model' necessitates also importing 'Use Case'. 'Impacts' dependency means that changing one element has a potential effect on the other. All of these dependencies should be modeled in the OPA only if they are valid in all possible process components where they will be used, e.g. 'Software Requirements Specification' will impact 'Software Architecture' in any imaginable process, thus the dependency is reusable.

4.3 The proposed organisation of an OSSP based on process components

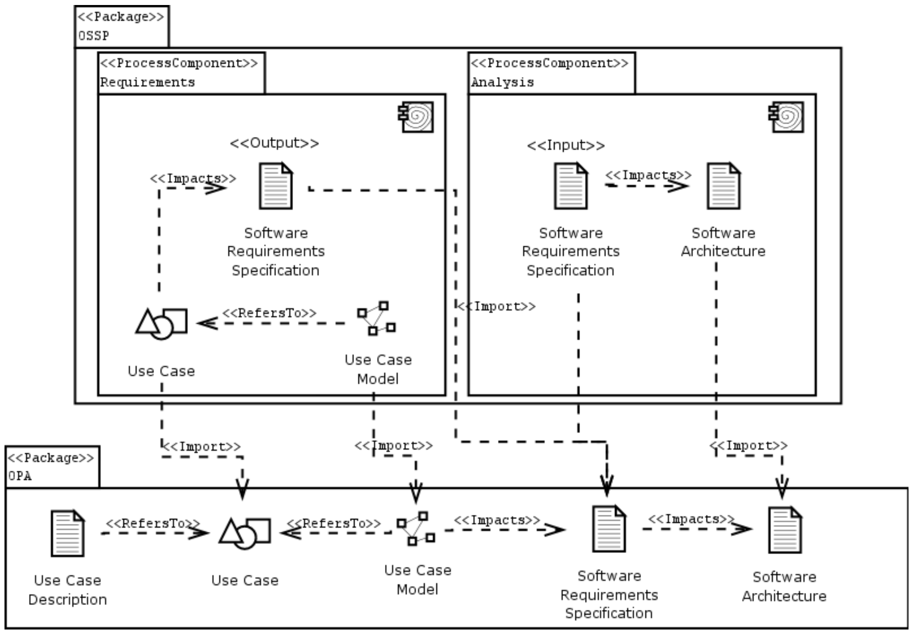

Fig. 4. The proposed organization process assets into OPA that contains low level reusable process elements, and OSSP that is the target of process authoring and contains goal-oriented small sized process components that package together the flow of activities and all related process elements.

Some of the process components in OSSP are going to be reusable and some not. Examples of reusable components are 'DefineSystemScope' or 'CreateCandidateArchitecture'. These components could have different versions for different types of projects, e.g. a light version of 'DefineSystemScope' for projects in familiar domain and a more detailed and more comprehensive one for new domains or multi stakeholder situations. Note that we propose to create process components based on clearly identified goals during the development work, rather than based on the structure of the process framework. As stated these components have variable degree of reusability. However this approach facilitates process authoring intermediate goals of development are a very natural view in process authoring.

Process authoring would then consist of dividing the project into clear intermediate goals, looking for reusable components for each identified goal, selecting the most suitable component depending on the particular project, tailoring or modifying the selected components, creating new components using elements from OPA where none could be found in the OSSP, and finally controlling the input and output work products to make sure that the components fit together. All of this involves a lot of manual work, but it seems that when practical issues are taken into account, more automation in process authoring is very difficult to achieve. The OPA of course gives high degree of reusability of process elements.

5 Discussion

In this ongoing work we aim at understanding the many faceted problem of decomposing software process models into reusable and easily pluggable components. Our original idea, inspired by guidance of SPEM standard, of creating large components using the structure of existing process frameworks seems to yield unmanageable process authoring task and no real reuse. Thus we started looking at smaller scale components and new ways of delineating process components. Some promising ideas have emerged, as reported above. However, there are many forces involved that must be taken into account. Processes with different life-cycle models and other process attributes probably require different modeling philosophy, e.g. document-driven methodologies using the traditional waterfall life-cycle model can be modeled using SPEM type language and process components quite easily while evolutionary methodologies with iterative activity flows are more demanding. Agile methodologies probably would require something very different because they usually lack clear intermediate work products and therefore process component interfaces could not be formed. The control relies heavily on inter-person communication and seamless integration of many activities by the developer into one whole. Establishing artificial interfaces and work products to implement the interfaces would be catastrophic in this environment. We believe that an agile process can not be decomposed at all into process components.

The traditional goals for OSSP are not easily achieved. The issue must be addressed from many viewpoints: process analysis, design, assessment, process component reuse, process improvement, process authoring, publishing, enactment and authoring. The issues are organization dependent; at least organizations size and business context affect the OSSP.

Common understanding on these issues must be achieved: academic community, tool vendors and standardization work all have a role here. SPEM standard version 1.1. has not gained widespread acceptance, possibly due to insufficient guidance on how it should be put into use. The standardization work for SPEM version 2.0. [9] is ongoing and hopefully succeeds better on these issues that have immense practical influence; is process modeling only going to be used for describing graphically the processes of the organization, or will we be creating a component oriented OSSP that is the target of SPI and can yield a tailor made process for each project, possibly supporting third party process component markets.

References

1. Object Management Group. *Software Process Engineering Metamodel Specification - Version 1.1*, January 5 2005. formal/05-01-06.
2. CMMI Product Team. Cmmi for systems engineering and software engineering (cmmi-se/sw, v1.1) - staged representation. Technical Report CMU/SEI-2002-TR-002, Software Engineering Institute, Pittsburgh, PA, USA, December 2001.
3. IBM. Ibm rational unified process. Software Product, 2005.
4. D. Wastell J.C. Derniame, B.A. Kaba, editor. *Software Process, Lecture Notes in Computer Science*. Springer-Verlag, 1999.
5. ISO/IEC, Geneva, Switzerland. *ISO/IEC 12207, Information technology - Software life cycle processes*, August 1 1995. ISO/IEC 12207:1995.
6. The frameworks quagmire. http://www.software.org/quagmire/. Accessed on May 30 2005.
7. Sarah A. Sheard. The framework quaqmire, a brief look. *Crosstalk*, September 1997.
8. Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Professional, February 4 1999.
9. Object Management Group. *Software Process Engineering Metamodel (SPEM) 2.0 - Request For Proposal*, November 4 2004. ad/2004-11-04.