TOWARDS A RIGOROUS PROCESS MODELING WITH SPEM

Benoit Combemale, Xavier Crégut
IRIT-LYRE
2, rue Charles Camichel, BP 7122
F-31071 Toulouse Cedex 7
{*benoit.combemale, xavier.cregut*}*@enseeiht.fr*

Alain Caplain, Bernard Coulette
GRIMM-ISYCOM
5, allee Antonio Machado
F-31058 Toulouse Cedex 9
{*caplain, coulette*}*@univ-tlse2.fr*

Keywords: Process engineering, Model engineering, SPEM, Semantics, OCL.

Abstract: Modeling software process is a good way to improve development and thus quality of resulting applications. The OMG proposes the SPEM metamodel to describe software processes. Its concepts are described through class diagrams. Unfortunately, it lacks a formal description of its semantics that makes it hard to use. So, we propose a specialization of SPEM that clarifies it and we use OCL to formally express constraints on it.

1 INTRODUCTION

A software application is a complex product which must be elaborated using a specific method : the *software development process*. The main aim of such a process is to guarantee the development of reliable software, conforming to their specification within expected deadlines and costs. Mastering the software appears to be a necessity because of the growing complexity and critical aspects of software. This must be done with the aim of obtaining better quality and improving reuse. The industrialization of software development began with methods like OMT and grew at the end of the 90's with unification proposals such as UML and RUP. Today, the last step is the use of Model Driven Engineering (MDE) for working on the complete lifecycle of software artifacts.

MDE (Bézivin, 2004) appeared with OMG proposals such as MOF (omg, 2002) and MDA (Miller and Mukerji, 2003). It tries to unify processes detailing with the crucial aim of reusing them. SPEM (Software Process Engineering Metamodel) (omg, 2005) is one of the OMG proposals. It is a metamodel for software development process specification that gives a rigorous syntax but only partly formalizes the semantics (thus implying some ambiguities), and does not offer any help on how to build a process model. It is why we have defined a specialization for its rigorous exploitation (Combemale, 2005). We first present the main concepts of SPEM (section 2) and then our specialization of SPEM (section 3). Before the conclusion, we present related works on section 4.

2 PRESENTATION OF SPEM

SPEM is a metamodel used for the specification of concrete software development process. It offers an object oriented approach using the UML notation. As an OMG proposal, SPEM is integrated in the pyramidal architecture of MDA organization as a MOF metamodel and as a UML profile (Breton and Bézivin, 2001). SPEM takes up a great deal of UML diagrams (packages, use cases, classes, activities, sequences, statechart), excludes some elements (node, component, etc.) (omg, 2005, §11.1) and adds several stereotypes that are presented hereafter (omg, 2005, §11).

SPEM is based on the idea (fig. 1) that a software development process is a collaboration between active abstract entities called *roles* which perform operations called *activities* on concrete and real entities called

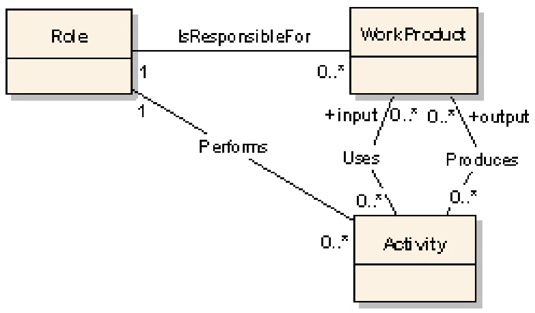

FIG. 1: SPEM conceptual model

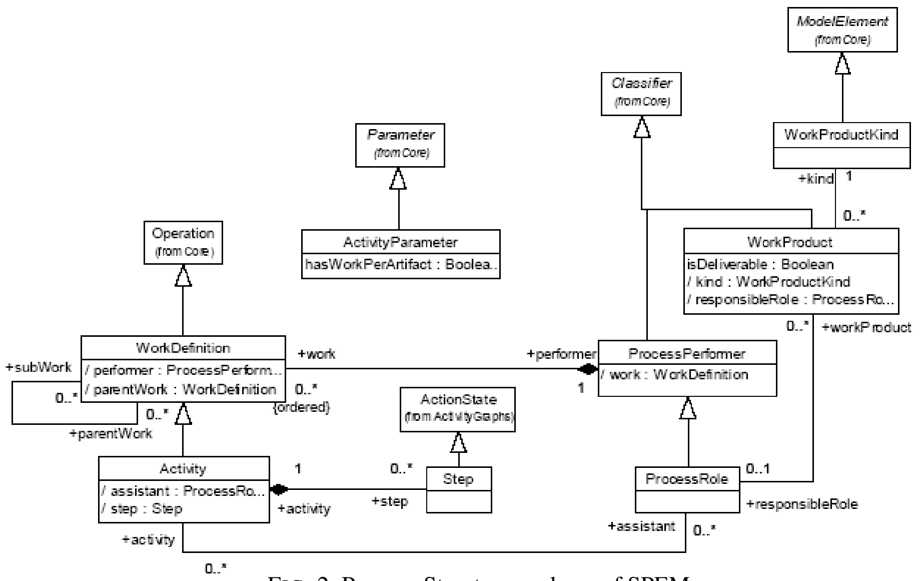

FIG. 2: Process_Structure package of SPEM

work products. The different roles act upon one another or collaborate by exchanging products and triggering the execution of certain activities. The objective of a process is to lead a set of products to a well defined state.

In fact, the SPEM metamodel is more complex. The extract given on fig. 2 proves it. SPEM defines the concept of *WorkDefinition* which can be decomposed reflexively. Beside the activities there are other specializations which are not drawn on fig. 2 : *Lifecycle* which is a sequence of *Phases* and *Iterations* to define a complex work. An *Activity* can be divided into *Steps*. A step is a specialization of *ActionState* (fig. 2) which implies a partial order (omg, 2004, §4.12).

Each *WorkDefinition* is under the responsibility of a unique role (*ProcessPerformer*). In the case of an *Activity*, a set of other roles (*ProcessRole*) can assist the main role for the realization.

SPEM also defines two important 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.

3 ADDING RIGOR TO SPEM

Using SPEM is difficult because the OMG proposal is very generalist and provides no directives on how to use it. Furthermore, its semantics is essentially expressed in natural language that leads to the construction of inconsistent process models because of the lack of a formal definition of concepts. For example, the *ProcessPerformer* is a concept known as being ambiguous (Bendraou et al., 2005).

So, we have decided to define a specialization of the SPEM metamodel whose purpose is to clearly define concepts and formally express their semantics with OCL (omg, 2003). Being more directive, our proposal brings more assistance in the industrial construction of a process and more facility in the use of SPEM. Thanks to the restrictions put on SPEM, it is possible to ensure the coherence of its models. Our proposal being a restriction of SPEM, our models conforms to SPEM.

3.1 SPEM metamodel specialization

Our metamodel (fig. 3) divides process models according to two main views. *The structural view* shows the process hierarchy : a process is associated to a lifecycle cut out in phases, themselves made up of activities (and iterations). *The descriptive view* details work definitions. It makes possible to classify them according to process roles and thus disciplines. Work definitions are specified through preconditions, goals and products used or realized.

As in SPEM, *WorkDefinition* metaclass is a concrete class. It allows to instantiate *WorkDefinitions* which will not be typed yet and thus not yet semantically defined in the process (e.g. analysis phase).

Because *ProcessPerformer* (fig. 2) has no clear semantics in the initial SPEM metamodel, we have merged it with *ProcessRole*. This merge avoids confusion

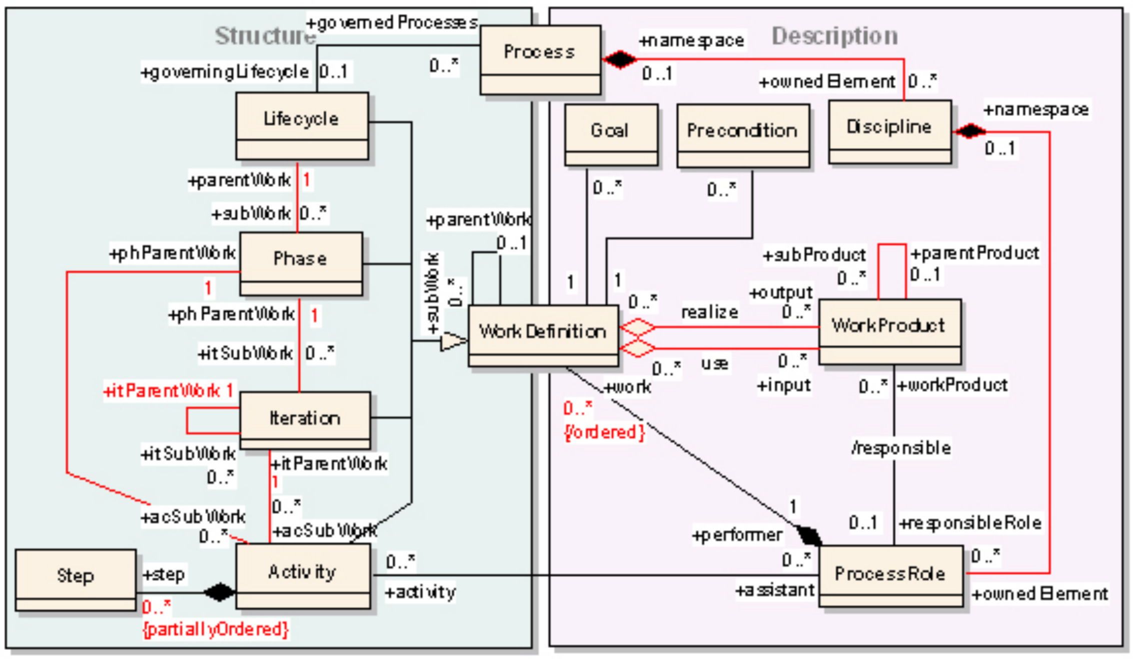

FIG. 3: Extract of our specialization of SPEM metamodel

between *ProcessPerformer* and *ProcessRole* by only defining *role* which is a set of capabilities.

In addition, most of the relations of our proposal are taken again from the original metamodel, either as is or by specialization. Thus, associations between *Lifecycle*, *Phase*, *Iteration* and *Activity* metaclasses fit to the redefinition of reflexive relation of the *WorkDefinition* metaclass of SPEM (fig. 2). We have constrained the multiplicity of the source of “0..*” by “1” (thus giving the composition semantics).

The relations which are not directly taken from SPEM metamodel are inherited or deduced from UML 1.4 (from the *Core* package, (omg, 2004, p.22)). For example, the reflexive relation on *WorkProduct* is a deduction of the relations that link *Classifier* metaclasses in UML. The “use” and “realize” relations between *WorkDefinition* and *WorkProduct* are also deduced from UML as shown on fig. 2. They allow to make explicit the relations described in the SPEM conceptual model (fig. 1).

3.2 Semantic details with OCL

Because syntactic added is not suffisant to semantically define the metamodel, we have used OCL (as recommended by the OMG) to add formal constraints. These constraints limit the possible instanciations and thus the valid process models. Here is an example of such a constraint : “A role must be responsible for all the products carried out by activities of which he is in charge, and reciprocally”.

```
context ProcessRole inv :
    let productsActivities : Set{WorkProduct} =
        – Definition of the WorkProduct set made by
        – the activities of which 'self' is responsible
        self.work→ select(a :WorkDefinition |
        a.oclIsTypeOf(Activity)).
        oclAsType(Activity).output→ asSet()
    in
        self.workProduct = productsActivities
```

Another OCL constraint formalizes the merge of *ProcessPerformer* and *ProcessRole* (sec. 3.1) in prohibiting the instanciation of the *ProcessPerformer* :

```
context ProcessPerformer inv :
    self.allInstances() → size() = 0
```

The static checks that we made are not sufficient and there should be extra dynamic checks. For example, it is impossible to ensure before enactment that an activity will respect its time limit but it can be checked during enactment. Furthermore, this kind of check is meaningful only during real project enactment and not during process model simulation. Let us note that a process model can not generally be completely defined before the project starts. If the great stages of the development are generally known, the detail of their realization depends on the difficulties encountered at the time of their realization. It could then be necessary to reinforce a team, to consider other solutions or another approach.

4 RELATED WORKS

The process modelling domain has been very active during the 90's. There was a strong separation between Process Description Languages (PDL) and process enactment engines. PDL could be classified into different categories : some of them were based on programming languages (ProcessWise/PML (Greenwood et al., 1992) or RHODES/PBOOL (Crégut and Coulette, 1997)), others on rules (MARVEL/MSL (Kaiser et al., 1988), ADELE/TEMPO (Warboys, 1994) or EPOS/SPELL (Warboys, 1994)), others on petri nets (SPADE/SLANG (Bandinelli et al., 1995)), and the others are hybrid solutions. Each of these languages had its own specific tools.

The actual tendancy is to unify PDL. Let us quote for example SPEM metamodel suggested by the OMG and XML-based languages like XPDL (*XML Process Description Language*) (WfMC, 2005) proposed by the WfMC(*Workflow Management Coalition*) or BPML (*Business Process Management Language*) (Arkin, 2002) proposed by the BPMI (*Business Process Management Initiative*). All these approaches define LDP's concepts (Breton, 2002) by proposing a syntax in the form of a metamodel for SPEM or a XML schema for XPDL and BPML. Semantics is only described informally (in natural language). Furthermore, process enactment is not formally defined even if there are specific engines for specific targets (e.g. BPEL4WS & BPEL-J).

5 CONCLUSION

In this article, we have presented our work on the modelling of software processes. Because the SPEM metamodel lacks rigorous and formal definition, we have proposed a restriction of SPEM that remains compatible with the standard and puts the focus on hierachical decomposition of *workdefinitions* (the *structural view*) and the categorization of process components (*roles*, *products* and *workdefinitions*) according to *disciplines* (the *descriptive view*). Semantics that are not graphicaly captured are expressed using OCL constraints either at the metamodel level or at the process level. Our SPEM specialization has been used to model a UML based method called MACAO (Combemale et al., 2006).

Unfortunately, OCL can only capture structural constraints. Our future work is to define an operational semantic for SPEM in order to enact a process model described in SPEM. So we are investigating several approaches including the ones that describe operational semantics for metamodel such as Kermeta, Xion and xOCL.

REFERENCES

(2002). *Meta Object Facility (MOF) 1.4 Specification.* OMG, Inc. Final Adopted Specification.

(2003). *Business Process Execution Language for Web Services v1.1.*

(2003). *UML Object Constraint Language (OCL) 2.0 Specification.* OMG, Inc.

(2004). *Unified Modeling Language (UML) 1.4.2 Specification.* OMG, Inc.

(2005). *Software Process Engineering Metamodel (SPEM) 1.1.* OMG, Inc.

Arkin, A. (2002). *Business Process Modeling Language.* Business Process Management Initiative.

Bandinelli, S., Fuggetta, A., Lavazza, L., Loi, M., and Picco, G. P. (1995). Modeling and improving an industrial software process. *IEEE Transactions on Software Engineering*, 21(5) :440–454.

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.

Breton, E. (2002). *Contribution à la représentation de processus par des techniques de méta-modélisation.* PhD thesis, Nantes University.

Breton, E. and Bézivin, J. (2001). Process-centered model engineering. In *5th IEEE International Enterprise Distributed Object Computing Conference*, Seattle, Washington, USA. IEEE Computer Society.

Bézivin, J. (2004). In search of a basic principle for model driven engineering. *CEPIS, UPGRADE, The European Journal for the Informatics Professional*, V(2) :21–24.

Combemale, B. (2005). Spécification et vérification de modèles de procédés de développement. Master's thesis, Toulouse II University, Master SLCP.

Combemale, B., Crégut, X., Caplain, A., and Coulette, B. (2006). Modélisation rigoureuse en SPEM de procédé de développement. In Lavoisier, H. S. ., editor, *LMO 2006*, pages 135–150.

Crégut, X. and Coulette, B. (1997). PBOOL : an object-oriented language for definition and reuse of enactable processes. *Software Concepts & Tools*, 18(2).

Greenwood, R. M., Guy, M. R., and Robinson, D. J. K. (1992). The use of a persistent language in the implementation of a process support system. *ICL Technical Journal.*

IBM and BEA (2004). *BPELJ : BPEL for Java.*

Kaiser, G. E., Feiler, P. H., and Popovich, S. S. (1988). Intelligent Assistant for Software Development and Maintenance. *IEEE software*, 5(3) :40–49.

Miller, J. and Mukerji, J. (2003). *Model Driven Architecture 1.0.1 Guide.* OMG, Inc.

Warboys, B. C., editor (1994). *Proc. of EWSPT'94*, volume 772 of *LNCS.* Springer–Verlag.

WfMC (2005). *Process Definition interface – XML Process Definition Language v2.0.*