A Comparison of Six UML-Based Languages for Software Process Modeling[1]

Reda Bendraou, Jean-Marc Jézéquel, *Member, IEEE*, Marie-Pierre Gervais and Xavier Blanc

Abstract— Describing and managing activities, resources and constraints of software development processes is a challenging goal for many organizations. A first generation of *Software Process Modeling Languages* (SPMLs) has appeared in the nineties but failed to gain broad industrial support. Recently however, a second generation of SPMLs appeared, leveraging the strong industrial interest for modeling languages such as the UML. In this article, we propose a comparison of these UML-based SPMLs. While not exhaustive, this comparison concentrates on SPMLs most representative of the various alternative approaches, ranging from UML-based framework specializations to full-blown executable meta-modeling approaches. To support the comparison of these various approaches, we propose a frame gathering a set of requirements for process modeling, such as *semantic richness*, *modularity*, *executability*, *conformity to the UML standard*, and *formality*. Beyond discussing the relative merits of these approaches, we also evaluate the overall suitability of these UML based SPMLs for software process modeling. Finally, we discuss the impact of these approaches on the current state of the practice, and conclude with lessons we have learned in doing this comparison.

Index Terms— Metamodeling, Process Modeling, Software Process Modeling Languages, UML.

1 INTRODUCTION

Since the late eighties, there has been a growing interest in viewing software systems as products resulting from the execution of orderly *software development processes* [1] [2]. While traditional verification and validation approaches (V&V) [3] have the goal of finding and removing defects in software products, *software processes* aim at documenting development practices that are empirically known to have an impact on software development time, cost, or quality [4].

Lonchamp defines a software process as "*the set of partially ordered process steps, with sets of related artifacts, human and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables*"[5]. This definition highlights the large number of factors that may influence the efficiency of software development processes. Software processes are often more complex and unpredictable than typical production processes, as they depend very much on people and circumstances. Activities of a software process do not all require automation and depend on communication, coordination and cooperation within a predefined framework [6].

Thus, a challenging goal for software development organizations is to find the means of rationally describing and managing activities, resources and constraints of their software development processes while taking into account all these characteristics. Once documented as process models, software development processes can become important assets of an organization. Process models can be used to reason about processes, to test and improve them to meet increasing quality and cost expectations. Beyond this *contemplative* use, process models can also be used in a more productive way to automate repetitive and non-interactive tasks.

The software community tried to answer the need for explicit process models with a wide range of *Software Process Modeling Languages* (SPMLs). Some of them were rules based (e.g., MARVEL) [7], others Petri net based (e.g., SPADE) [8] or programming languages based (e.g., SPELL, APPL/A) [9] [4]. While these first-generation languages were executable and put a strong emphasis on formality, they did not gain much attention from the industry [10]. Their complexity, their use of low-level formalisms and their inflexibility were among the causes of their limited adoption [11].

Another factor that contributed to their low impact was the multiplicity of formalisms and proprietary notations used as support for describing software processes [12]. The continuing proliferation of these first-generation SPMLs has naturally raised the need for standardizing software process descriptions. Instead of reinventing the wheel, many industry and research teams were appealed by the wide diffusion of the UML (Unified Modeling language) and explored the possibility of using it as a process modeling language. UML is indeed a standard providing a rich set of notations, diagrams, and extension mechanisms. Whatever its advantages and drawbacks, it is undeniably one of the most adopted modeling lan-

- *R. Bendraou is with UPMC (University of Pierre & Marie Curie), 4 place de Jussieu F-75005, France. E-mail: reda.bendraou@Lip6.fr.*
- *J.M.. Jézéquel is with the INRIA-Rennes Bretagne Atlantique, Campus de Beaulieu, Rennes F-35042, France. E-mail: Jezequel@inria.fr.*
- *M.P. Gervais. is with the University of Paris X, Nanterre F-92001,France., E-mail: Marie-Pierre.gervais@lip6.fr.*
- *X.Blanc is with the UPMC university, 4 pl. Jussieu, Paris 75005, France. E-mail: Xavier.blanc@lip6.fr.*

1 This work has been partially supported by the S-Cube Network of Excellence of thq European FP7 and the IST Modelplex project, contract n° IST-3408.

guage of this decade. As a result, many UML-based approaches for software process modeling in particular and for process modeling in general emerged. Some of them succeeded even to be standardized[13], [14], raising again the usual questions: *Does UML offer a real benefit as a basis for a process modeling language? If I have to adopt a UML-based approach, which one fits best my needs*? *What would be the cost of adopting such approaches?*

Even if in the literature many works addressed the use of UML within the several phases of the software development process, none of them gives a detailed review of approaches that use UML as the basis for the definition of a software process language. By comparing UML-based languages for software process modeling, this paper aims to help software engineers answer the above-cited questions. Our goal is to provide project managers, methodologists and process modelers with a detailed review of these approaches, highlighting their advantages and their drawbacks. Since each reviewed approach offers specific features, this comparison should help in choosing the appropriate language depending on project specific needs (e.g., documenting processes, support for enactment and simulation, improvement, process compositions, etc.).

The approaches addressed by this comparison are the OMG standard SPEM (version. 1.1 & 2.0), the PROMENADE language, Di Nitto et al. approach, Chou's approach and the UML4SPM language. They regroup representative initiatives from the industry, research projects and academic research groups. In order to compare them, we propose a frame gathering a set of requirements for Process Modeling languages (PML), as identified by several research efforts in the literature. These requirements are semantic richness, executability, modularity, formalization, tooling support, graphical representations and the support of multiple perspectives and the conformance of the reviewed approaches with regard to the UML.

Before presenting this comparison, the next section addresses one of the preliminary questions this paper tries to answer: *is UML suitable for process modeling?* We will see for instance that while UML presents a serious potential in terms of expressiveness, the executability and formality aspects remain one of its major weakness. In Section 3, we briefly introduce the PML requirements we used for comparing the above-cited approaches in Section 4. Additionally, Section 4 discusses how the various approaches deal with the executability and formality issues. The impact of these approaches on the current state of the art is discussed and some lessons learned while comparing them are given in Section 5. We also give the reader the means to answer the question: *which approach fits best my needs*? Finally, Section 6 concludes this paper.

2 The suitability of UML for process modeling

In the last decade, UML succeeded to become the *de facto* standard for modeling software systems. However does this necessarily make it a good candidate for modeling processes, including software and business processes?

Arguably, UML offers a powerful set of notations and diagrams that allow capturing both static and dynamic aspects of processes and can increase their understandability. Most widely used diagrams in the context of process modeling are (i) the class diagrams for representing process constituents and the relationships that link them, (ii) the state machine diagrams for modeling possible workproduct or activity states and the events that trigger state changes, (iii) finally, the activity diagrams for modeling the workflow.

Furthermore, in its version 2.0, UML goes beyond graphical representations by offering a high potential for expressing a large variety of processes. Thanks to *Activity* and *Action* packages, it provides concepts for expressing proactive and reactive controls, conditional branches, loops, exception handling as well as a numerous actions with computational semantics. It also supports a large number of *Workflow* patterns, a taxonomy of generic, recurring constructs originally devised to evaluate workflow systems, and more recently used to successfully evaluate business process languages and Process Aware Information Systems (PAIS) in general (see [15], [16] and [17]). In accordance with Jablonski and Bussler's original classification [18], these patterns span the *control-flow*, *data* and *resource* perspectives of PAIS, the two later perspectives being more specific to business processes rather than to software processes. In [19], authors evaluated the capacity of UML2.0 in modeling twenty control-flow patterns that commonly recur in process models. Examples of such patterns are parallel split, multiple merge, deferred choice, etc. UML2.0 succeeded in representing all of them except for four, which makes it more expressive than some business process formalisms (e.g. BPEL: Business Process Execution Language) [20]. Data patterns mainly deal with data visibility, data interaction and data transfer and routing. Examples of such patterns are the multiple instances data pattern, the database task trigger patterns and so on. In [21], it has been demonstrated that eighteen of the forty data patterns were supported by UML2.0, which remains quite satisfactory. As for resource patterns, they address all the issues about work allocation to process's resources, the ability for resources to see the work status, resources allocation conflicts, work distribution and so on. According to [22] however, UML2.0 only satisfies six of the forty-three resource patterns, which reduces its suitability for modeling the resource perspective. Still many of these perspectives can be addressed at a lower level by an execution support of UML-based process models.

Regarding executability, it is clear that from the hypothetical day when a UML virtual machine would be universally adopted, UML-based process models would have a real benefit. Process modelers supposed to be already familiar with UML diagrams would then simply have to draw their process models using their usual UML tools. They would then be able to test, execute and debug them as everyone does with her usual programming language. UML2.0 offers the potential to define such virtual machine thanks notably to the *Activity* and *Action* packages, which come with an operational semantics. Some ambiguities in this operational semantics (given in natural lan-

guage in the standard), have however to be first fixed. This is one of the purposes of a recent initiative at the OMG, called *Executable UML Foundation* [23]. The objective of this proposition is to unify the definition of a computationally complete and compact subset of UML 2.0 (the "Executable UML Foundation") with a full specification of the execution semantics of this subset. "Computationally complete" means that the subset shall be sufficiently expressive to allow definition of models that can be executed on a computer either through interpretation or as equivalent computer programs generated from the models through some kind of automated transformations. We will see in this paper how one of the reviewed approaches (i.e., UML4SPM) get inspired by the Executable UML initiative in order to propose a process engine for executing UML-based process models. Other approaches like Di Nitto's one proposes to generate executable code from UML diagrams used for modeling the software process. Along the same line, some academic efforts and industrial projects already tried to formalize and to execute UML2.0 Activities [24], [25].

The lack of formality is clearly a weakness of the UML language. However, as discussed in section 3.5, this problem can be mitigated by relying on the formal semantics of some other well-known formalism. In the context of process modeling, Activity diagrams are the most used UML diagrams for modeling the process flow of work. A definition of their formal semantics was already provided by many works in the literature [26], [27] and [28]. Most approaches consisted in mapping activity diagram concepts into Petri nets in order to perform model analysis and verification.

In Section 4 we will see how the compared approaches use the UML potential as a basis for process modeling and how they extend it in order to overcome some of its limits.

In the next section we introduce the PML requirements used for the SPMLs comparison.

3 Process Modeling Language Requirements

Many requirements related to process modeling languages have been identified in the literature [29] [30] [31] [32] [33]. They vary from facilitating human understanding, to analyzing processes, or to providing an automated execution support. For carrying out the comparison of UML-based SPMLs, we selected most predominant and common requirements from the above-cited references. In the following, we introduce them briefly; the interested reader can refer to the referenced papers for more details. We also consider the tooling support and conformity of the compared approaches to the UML standard as core requirements.

3.1 Semantic Richness (Expressiveness)

Semantic richness relates to a SPML ability to express what is actually performed during software development processes. It encompasses many aspects summarized as follows:

Core Process Elements

Early classification of the constituents of software process models have been proposed in the literature [34] [35] [36] [5, 37] [38]. This classification considers as core process elements the concepts of *Activity*, *Artifact*, *Role and Agent*.

Activities and Actions Coordination

These mechanisms fall into two categories, *Proactive Control* and *Reactive Control* [39]. Proactive control is an imperative specification of the order in which activities have to be executed. Main means used to formalize proactive control are *Precedence* relationships (i.e., start-start, start-finish, finish-start, finish-finish) [40] and *Call Actions* (i.e., explicitly calling an activity, an operation, etc.). Reactive control is the specification of the conditions or events in response to which activities are to be executed. Examples of means used to express reactive control are Exceptions and Event handling,.

Exception Handling

Exceptions are parts of processes. They can result from violations of some process constraints or steps and can include changes in resources, organizational structure, task priority, and so on [41]. Thus, the process language should be able to capture possible process failures and to propose strategies to resolve them and to recover a stable state of the process.

3.2 Conformity to the UML Standard

In this aspect, we aim to evaluate to what extent the compared approaches reuse the UML standard. In other terms, do the approaches define an extension in form of a MOF metamodel, of a Profile, or simply reuse UML diagrams as a base framework for their language. This would allow for instance for a tool editor to evaluate the cost of building a tool to support such approaches and for process users to evaluate to what extent they can reuse or customize their favorite UML tool.

3.3 Graphical Representations and Support of Multiple Views

In this point, we consider the clarity of process modeling and the support of multiple perspectives/views on the process. For the first point, we use an actual real example (see below) of a software development process and model it with each of the compared approaches. For the second one (i.e., support of multiple views), we enumerate the multiple views offered by each approach and we check whether they are mutually consistent.

Process Example

This process example was provided by our industrial partners within the IST European Project MODELPLEX. It is first described in natural language and then modeled using the various approaches

The process is composed of two phases: "*Inception*" and "*Construction*". For brevity reasons, the "*Construction*" phase is skipped here. The "*Inception*" phase is composed of two activities: The "*Elaborate Analysis Model*" activity and the "*Validate Analysis Model*" activity. The "*Elaborate Analysis Model*" activity takes as input a requirements document and produces a UML "Analysis Model". The "Analysis Model" is then taken as input by the "*Validate*

Analysis Model" activity that is composed of the following steps: 1) Get the "Analysis Model"; 2) Submit the UML model for validation to a UML Checker Tool, which emits a validation report; if the "Analysis Model" is valid then go to the next phase. If the "Analysis Model" is invalid then, comeback to the "Elaborate Analysis Model" activity. The role in charge of both activities of this phase is played by the "Analyst".

3.4 Executability

Having a support for executing process models can help in coordinating between process's participants, enforcing artifacts routing, ensuring constraint integrities and process deadlines. It can also be of an effective aid since process models can be used for simulation and testing.

3.5 Modularity

Modularity is about the ability to combine different chunks of processes in order to build a new one.

3.6 Formality

In this paper, since the SPMLs we compare are UML-based, we rely on the literature [42] [43] [25] [44] to assess the formality of UML. We also discuss the different approaches adopted by these SPMLs, for verifying, validating or analyzing process models.

3.7 Tooling Support

We also consider also the tooling support criterion, which is of prime importance when having to choose between different SPMLs.

4 Comparing UML-Based Software Process Modeling Language

In this comparison, we focused on representative SPMLs from four different kinds of approaches.

- Standard SPMLs, i.e. the various version of OMG's SPEM (Software Process Engineering Metamodel) that come both in the form of a MOF meta-model and a UML profile (sections 4.1 & 4.2).
- SPMLs resulting from the specialization of an Object-Oriented framework, with a modeling layer in UML and an implementation layer partially generated from the UML layer (DiNitto's approach [10] and PROMENADE [45] [46], section 4.3).
- SPMLs consisting of high-level UML-Based diagrams and an ad-hoc low-level process language (Chou's approach [47], section 4.4).
- SPMLs trying to complement the meta-level approach with full executability (UML4SPM, section 4.5).

4.1 Standard UML-based SPMLS

4.1.1 SPEM1.1 Evaluation

SPEM1.1 (Software Process Engineering Metamodel) is the OMG's standard for software process modeling. It was adopted by early 2005 [14], [48]. Even if SPEM2.0 is under finalization at the OMG, we found it interesting to present SPEM1.1, since some companies have already adopted it and may be interested to evaluate the cost and benefits of migrating SPEM 1.1 models to SPEM 2.0.

4.1.1.1 Semantic Richness

Table 1., summarizes expressiveness aspects of SPEM1.1.

Approach	SPEM1.1
Semantic Richness	
Core Process Elements	
Activity	Work Definition, Activity. An Activity may be composed of *Steps*
Role	Process Role
Artifact	WorkProduct
Agent	**Not addressed**
Tool	**Not addressed**
Activities Coordination	
Proactive Control	Kinds of precedence ensured: *start-start*, *finish-start* or *finish-finish*. The *start-finish* precedence is lacking
Reactive Control	**Not addressed**
Exception Handling	**Not addressed**

Table 1. Expressiveness aspects of SPEM1.1

4.1.1.2 Conformity to the UML Standard

SPEM1.1 uses some basic modeling concepts from UML1.4 to describe rules, constraints, vocabulary, and notation to be used in defining process models [49]. It comes in form of a MOF 1.3-compliant metamodel and a UML1.4 profile. The metamodel is defined as an extension of a subset of UML1.4.

4.1.1.3 Graphical representations and support of Multiple Views

SPEM1.1, through its UML profile, proposes to reuse a subset of UML diagrams (i.e., Class, Activity, Statechart, Sequence, Use Case, and Package diagrams) in order to describe process aspects. These diagrams are customized thanks to a set of graphical icons, which represents language's concepts (e.g., WorkDefinition, Step, WorkProduct, etc.). Mainly, one needs to handle the Class diagram for representing relationships between different process constituents, the Activity diagram to describe the sequencing of activities with their inputs and outputs and finally, the Use Case diagram to show the relationships between process roles and the main work definitions.

Example of a process representation using SPEM1.1

In (fig. 1), the workflow view of the process example - the inception phase- introduced in section 3.2 is given using SPEM1.1 notations. SPEM1.1 uses UML1.4 activity diagrams, which are customized by icons defined in the SPEM1.1 profile. Swim lanes are used to define process roles. One has to notice that an activity cannot be shared by more than one role (one role by swim lane according to the UML standard). The details of the "Validate Analysis" activity were skipped for brevity reasons.

4.1.1.4 Executability

Executability is out of SPEM1.1 scope.

4.1.1.5 Modularity

One of the major issues of SPEM1.1 is *ProcessComponent* compositions, which is supposed to be the mechanism for process compositions. A *ProcessComponent* is a chunk of process description that is internally consistent and may be reused with other *ProcessComponents* to as-

semble a complete process. However, developers who want to combine two or more *ProcessComponents* in order to get one coherent process, have to carry out a manual *unification procedure* (i.e. renaming process elements). Indeed, to combine for instance two *ProcessComponents* P1 and P2, at least the output *WorkProducts* from P1 must be unified i.e., made identical with the *WorkProducts* inputs to P2. Other elements may possibly be unified in addition, such as *ProcessRoles*.

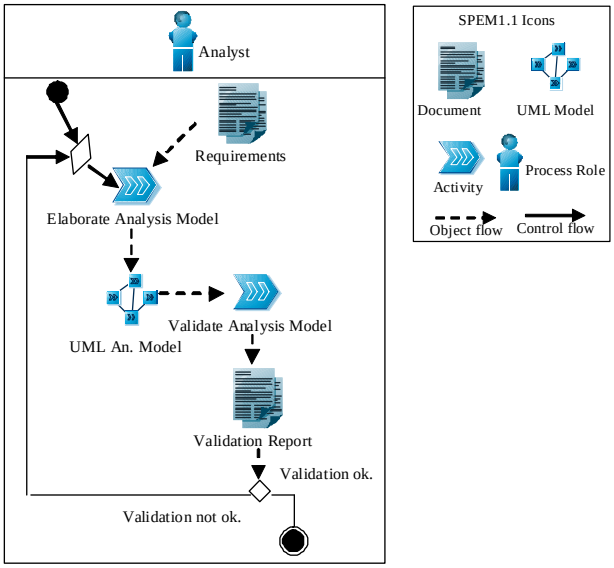


Figure 1. Representing a process example using SPEM1.1 notations.

4.1.1.6 Formality

Works dealing with formality in SPEM1.1 mainly attempt to address the numerous ambiguities and inconsistencies of the language by constraining the MOF metamodel. In [50], many of these ambiguities are identified. In [51], additionally to the identification of other inconsistencies, authors propose the use of OCL for a more rigorous specification of SPEM1.1's semantics.

4.1.1.7 Tooling Support

Regarding the tooling support, we can mention the Rational Process Workbench (RPW) from Rational [52], IRIS Suite from Osellus [53], and SPEM Profile from Objecteering [54]. However, each of these tools proposes its own formalism for process model persistency. Thus, no model exchanges are possible between the various tools.

Discussion

To summarize, SPEM1.1 presents the advantage of being based upon UML, which makes it a good candidate for a large adoption since many people are familiar with the UML. However, SPEM1.1 has had a limited success within the industry. One of the obstacles was that the standard included many ambiguities. Another one was because SPEM1.1 process models were only contemplative models. No execution support was provided.

4.1.2 SPEM2.0 Evaluation

SPEM2.0 comes with a new attractive vision. It consists in separating all the aspects, contents and materials related to a software development methodology from their possible instantiation in a particular process [55].

SPEM2.0 defines three compliance points. The first one called "*SPEM Complete*" is dedicated to case tool providers who want to support the description of large-scale method libraries as well as process definitions that may reuse the method library contents. It contains all SPEM2.0 packages. The second compliance point is the "*SPEM Process with Behavior and Content*" and is dedicated to tool providers who are only interested in providing concepts for describing process models without referring to a particular method library (e.g., the Agile community). The last compliance point is the *"SPEM Method Content"* ; it is recommended for implementers who primarily focus on managing the documentation of development methods, techniques, and best practices.

4.1.2.1 Semantic Richness

Table 2., summarizes expressiveness aspects of SPEM2.0:

Approach	SPEM2.0
Semantic Richness	
Core Process Elements	Depends on the Compliance point used
Activity	Activity, Task Definition
Role	RoleUse, Role Definition
Artifact	WorkProductUse, WorkProduct Definition
Agent	**Not addressed**
Tool	Tool Definition
Activities Coordination	
Proactive Control	Ensured thanks to the *WorkSequence* concept. Kinds of precedence: *start-start*, *finish-start*, *finish-finish* and *start-finish*
Reactive Control	**Not addressed**
Exception Handling	**Not addressed**

Table 2. Expressiveness aspects of SPEM2.0

4.1.2.2 Conformity to the UML Standard

SPEM2.0 comes in the form of MOF2.0-compliant metamodel that reuses UML2.0 Infrastructure [56] and UML2.0 Diagram Interchange specifications [57]. No concept from the UML2.0 Superstructure (Sp) [58] is reused. The standard comes also in form of a UML Profile where each element from the SPEM2.0 metamodel is defined as a stereotype in UML2.0 Sp.

4.1.2.3 Graphical representations and support of Multiple Views

Since SPEM2.0 does not reuse UML2.0 Sp., it does not rely on the diagrams offered by UML. Instead, SPEM2.0 proposes a behavioral model for describing the workflow of the process using a set of icons defined in the SPEM2.0 profile. It also proposes kinds of proxy classes in order to link the process description with some external behavioral models defined in other formalisms such as BPMN [59], UML activity diagrams, etc. However, the standard is not clear about how to achieve this and states that this remains a tool's implanter responsibility. If many external behavioral models are used, it is up to the process modeler to ensure consistency between these models.

Example of a process representation using SPEM2.0

In SPEM2.0, different concepts are proposed for representing process elements. Their use mainly depends on the compliance point chosen by the process modeler for

modeling the process. For instance, if one is using the *"SPEM Process with Behavior and Content"* compliance point, she would be referring to artifacts used by process's activities as *"WorkProduct Uses"*. However, when using the *"SPEM Method Content"* compliance point and one is describing a method, she would refer to artifacts produced or consumed by a method's *Task Definitions* as *"WorkProduct Definition"*. Finally, if one is using *"SPEM Complete"*, she can use both *WorkProduct Use* and *WorkProduct Definition*. The *WorkProduct Definition* would be used for documenting the artifact used by her method and the *WorkProduct Use* would be used in her process model as a reference (pointer) to the *WorkProduct Definition* given in the instantiated method instead of describing again the artifact within the process model.

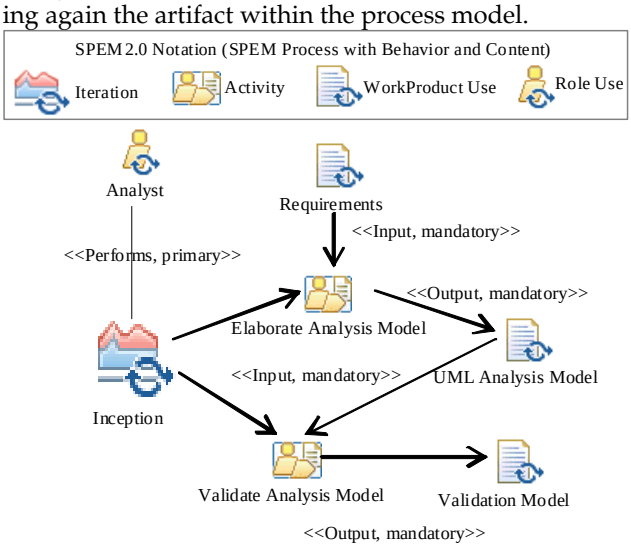


Figure 2. Representing a process example using SPEM2.0 notations.

Figure 2. presents the description of the process example using the *"SPEM Process with Behavior and Content"* compliance point. Notice that using another compliance point, would imply the use of a different notation. The last alternative consists in using an external behavioral model such UML activity diagrams. In this case, the process representation would roughly resemble the one given in figure 1 (cf. section 4.1.1.2).

4.1.2.4 Executability

Even if process enactment was among the main requirements when the SPEM2.0 RFP was issued [60], the adopted specification does not actually address the enactment issue. Nevertheless, it clearly suggests two possible ways of enacting SPEM2.0 process models. The first option consists in mapping SPEM2.0 process models towards some project planning tools such as IBM Rational Portfolio Manager [61] or Microsoft Project [62]. While this approach maight be very useful for project planning it is not considered as process enactment. Project plans are used by project manager in order to estimate whether the process would be in schedule, whether more persons need to be assigned to process tasks, etc. There is no support for automatic task assignments to responsible roles, no automatic routing of artifacts, no automatic control on work product states and so on. The second option proposed by the standard is to transform SPEM2.0 process models into some business process execution formalisms such as BPEL [20]. Nevertheless, SPEM2.0 supposes that this task is tool implementer's responsibility. In [63], authors demonstrate the limit of such approach.

4.1.2.5 Modularity

SPEM2.0 provides various mechanisms for reusing, extending and customizing process models and methods. At the *"SPEM Process with Behavior and Content"* compliance point, extension of process models is ensured thanks to the *Activity*'s *"Activity Use Kind"* property (enumeration). Depending on the value of the property, a process's activity 1) can extend an activity from another process; 2) can be extended by another process's activity or 3) completely replaces an activity in another existing process.

At the *"SPEM Method Content"* or *"SPEM Complete"* compliance points, the specification proposes mechanisms such as *Variability Element* and *Process Component*. The former allows not only for extending process's activities as in the *"SPEM Process with Behavior and Content"* compliance point but also to any metaclass inheriting the *Variability Element* abstract metaclass. This would allow a process model or contents of one method to redefine, reuse or replace another method's contents or process models. The detail of this mechanism is given in more detail in [55]. The latter, i.e., *Process Component*, is a means for defining a kind of reusable black box process identified by its ports (i.e., *Workproducts* inputs and outputs of the *Process Component*). Finally, the concept of *Method Plugin* is introduced. It makes it possible to organize method contents and processes in one independent and reusable plugin. A *Method Plugin* can extend many other *Method Plugins* and can be extended as well.

4.1.2.6 Formality

To deal with formality in SPEM2.0, some references propose to use a translational semantics approach. The approach consists in relying on a well-defined formalism to express semantics of a given language [64]. This enables the validation of SPEM2.0 process models by using model-checking techniques. For example, [65] used timed Petri nets to formally express SPEM2.0 process models. LTL (Linear Temporal Logic) formulae related to process model properties are then generated over the Petri nets and passed to a model-checker[2]. This allows checking some properties on the process such as: does every process's activity eventually start? Do all started activities eventually finish? Etc.

4.1.2.7 Tooling Support

Regarding the tooling support, an open source project called EPF (Eclipse Process Framework) [66], which was initially introduced for supporting the IBM's UMA method (Unified Method Architecture), is on the way to be fully compatible with SPEM2.0. A commercial version of this tool exists: the Rational Method Composer (RMC) tool [67]. Objecteering also proposes a commercial tool on top of both EPF and Microsoft Project called PRO3 [54]. Tool vendors aiming at a SPEM2.0 profile implementation

[2] TINA Model-Checker (TIme petri nets Analyser) at: http://www.laas.fr/tina/

still have to face the problem of defining the OCL constraints for the UML2.0 metamodel in order to respect the SPEM2.0 semantics. Indeed, the specification defined the profile but intentionally left the writing of OCL rules up to the profile implementers. The argument was that the semantic was already defined in the SPEM (MOF) metamodel.

Discussion

The main advance in the SPEM2.0 specification is the proposition of a clear separation between the contents of a method and its possible use within a specific process. However, with the introduction of extension mechanisms, compliance points and the notion of *Method Plugins*, the specification turns out to be very complex and hard to understand. Regarding executability, we saw that SPEM2.0 does neither provide concepts nor formalisms for executing process models. For modularity aspects, the standard proposes powerful mechanisms for extending process models and methods, which requires extensive implementation efforts in order to respect the semantics of all the proposed extension mechanisms.

4.2 Framework Specialization Approaches

In this Section, we investigate SPMLs resulting from the specialization of an Object-Oriented framework, with a modeling layer in UML and an implementation layer partially generated from the UML layer. We principally focus on DiNitto's approach [10], and then we briefly overview a variant of this approach called PROMENADE [45] [46].

4.2.1 Di Nitto et al. Approach Evaluation

Di Nitto's et al. approach [10], aims at assessing the possibility of using a subset of UML1.3 [68] as an executable PML. It comprises two main phases. The first one consists in describing processes using UML diagrams. The second phase consists in translating these UML diagrams into code that can be enacted by the team's events-based workflow engine called OPSS (ORCHESTRA Process Support System) [69].

4.2.1.1 Semantic Richness

This approach uses UML1.3 diagrams as a high-level modeling language. There is no extension to the UML1.3 metamodel, no stereotyping or new concepts introduced. The approach uses a predefined class diagram to represent basic process constituents such as *Activity*, *Artifact*, *Agent*, etc. Table 3., summarizes expressiveness aspects of Di Nitto's approach:

Approach	Di Nitto's et al.
Semantic Richness	
Core Process Elements	provided in terms of UML1.3 classes (Instances of the UML *Class* metaclass)
Activity	Activity
Role	**Not addressed**
Artifact	Artifact
Agent	Human Agent
Tool	Software Agent
Activities Coordination	
Proactive Control	Use of UML1.3 sequence control flow for modeling *finish-start* precedence and *Join*, *Fork* for modeling *start-start*
Reactive Control	In state diagrams, events are used as means to trigger transitions allowing activity state changes
Exception Handling	Possibility to define error states in state diagrams

Table 3. Expressiveness aspects Di Nitto's approach

4.2.1.2 Conformity to the UML Standard

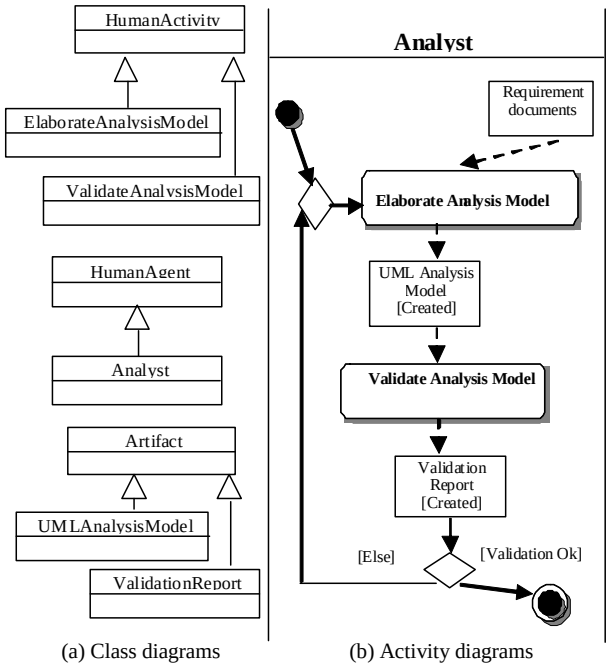
In this approach, process constituents are modeled as a specialization of a predefined UML1.3 class diagram that comprises classes used by the OPSS engine. These classes can be specialized by process modelers in order to adapt the predefined class diagram to a specific process. A process modeler willing to use OPSS has to start defining her own activity types, agent types, etc. by specializing the existing classes.

4.2.1.3 Graphical representations and support of Multiple Views

The UML diagrams used in this approach are activity diagrams for modeling the flow of work (called *Activity Graphs* in UML1.3), class diagrams to associate process concepts with concepts that are part of the OPSS engine. Each OPSS class has an associated state machine to describe the lifecycle of its instances. A precise and complete definition of these state machines is critical for process execution, since they encapsulate the process business rules. However, one has to notice one lack. There is no link between the activity and class diagrams. The approach does not define one activity diagram per class but one activity diagram for all classes representing an activity in the class diagram. Each class is represented by an action or a subactivity state in the activity diagram. Authors claim that the link between action or subactivity states defined in the activity diagram and classes defined in the class diagram is checked through name matching. However, this may work for one simple class diagram, but in case of combining many class diagrams to form one global process definition this may become complicated.

Example of a process representation using Di Nitto's approach

In order to represent a process using Di Nitto's approach one has 1) to specialize OPSS classes given in the predefined class diagram (fig. 3a), 2) to model the flow of work using activity diagrams (fig. 3b) and 3) to model state machines proper to each activity to define process business rules (not represented). Authors claim that using the composition relationship in the class diagram allows defining activities composed of many other activities, each activity having its own performer. However, this composition aspect cannot be reflected in the activity diagram. In the activity diagram, an action state (atomic action) or a subactivity state is realized by one and only one role materialized by the swim lane. Thus, representing a compound activity with each of its component activities having a different performer is impossible with UML activity diagrams.



(a) Class diagrams (b) Activity diagrams

Figure 3. Representing a process example using DiNitto's approach.

4.2.1.4 Executability

To execute the process, all diagrams are used for generating the code. User-defined classes derived from predefined classes to describe specific elements of the process are translated into the corresponding Java classes, with their attributes, methods and associations as described in the UML class diagrams. The body of methods is defined according to the information provided by the corresponding state and activity diagram. However, the weak point in this translation is how precedence relationships (sequencing of activities) defined in the activity diagram are reported into the Java code. Unfortunately, the only reference to this by the authors is: *"relations represented in the activity diagrams are translated into Java code which manages such relations"* [10]. Another lack of this approach in terms of executability is that the code of new operations introduced in user-defined classes can only be inferred from the state diagram of the class. If the modeler does not provide enough information in the state diagram, the implementation of the operation is left incomplete.

4.2.1.5 Modularity

Modularity was not addressed in this approach.

4.2.1.6 Formality

No approach is proposed for validating the process model. An operational semantics is given to process models consisting of class, activity and state diagrams, through a Java code generation.

4.2.1.7 Tooling Support

Authors propose a code generator that transforms UML diagrams into Java code that is used as input of the OPSS workflow tool developed by the same team [69].

Discussion

The advantage of this approach is that process constituents can easily be defined by simply specializing a set of predefined classes provided by the approach in form of a UML class diagram. The flow of work is given in activity diagrams, while the lifecycle of each entity is defined by a state machine. However, we saw that the activity and class diagrams have no links with each other. The approach does not extend the UML language nor introduce new concepts. Process elements are simply instances of the UML *Class* metaclass, which means that they all have the same semantics and notation as the UML *Class* metaclass. Regarding execution, it is mostly based on how state diagrams defined by the user are precise enough and sound in order to enable a complete code generation and to allow process execution within OPSS. Otherwise, code has to be added manually. The weak point in the executability aspect remains how information defined in activity diagrams, state machines and class diagrams are integrated to generate the Java classes needed for the execution. Authors did not detail how this integration is realized. Modularity was not addressed by the approach.

4.2.2 The PROMENADE Approach Evaluation

Promenade stands for (**PRO**cess-oriented **M**odelling and **EN**actment of software **DE**velopment). It is a software process modeling language defined in the context of the ComProLab project [45], [46] and is based on UML1.3 [68]. Since this approach follows the same principle as DiNitto's, we briefly present its main features, and we discuss its advantages and drawbacks regarding PML requirements.

To model a process using Promenade, one has to specialize the set of predefined classes provided by the approach. To define precedence between process's tasks, one has to define a precedence graph, which defines the order between all tasks of the process. Precedence rules are described using a declarative formalism, which is quite simple. However, authors do not specify how the precedence graph (including precedence rules) is to be integrated with the class diagram to form a complete process description.

For combining many processes (modularity), authors propose to make all process elements of the same type from the various processes, inherit from the same predefined classes defined by Promenade. However, to avoid name clashes, authors propose to rename manually all the classes that represent the same artifacts but that are called differently from one process into another. What makes the approach more complex is that the precedence graph has also to be modified accordingly. Finally, the approach does not provide any mechanism or way to execute Promenade process models; no tool was provided.

4.3 Two Layers Approaches

In the context of a research project financed by the National Science Council of Taiwan, Chou proposed a software process modeling language consisting of high-level UML1.4-based diagrams and a low-level process language [47]. While UML diagrams are used for process's participants understanding, the process language is used to represent the process - from UML diagrams – in a machine-readable format i.e., a program.

4.3.1 Chou's Approach Evaluation

4.3.1.1 Semantic Richness

Table 4., summarizes expressiveness of the approach:

Approach	Chou's Approach
Semantic Richness	
Core Process Elements	provided in terms of UML1.4 classes (Instances of the UML *Class* metaclass)
Activity	Activity
Role	**Not addressed**
Artifact	Document
Agent	**Not addressed**
Tool	Tool
Activities Coordination	
Proactive Control	Ensured thanks to UML *Activity sequence* for modeling finish-start. Use of *fork* and *join* elements for modeling *start-start*
Reactive Control	Use of events and exception handlers
Exception Handling	Exception Handlers are represented as activities in AD and as operations in the code

Table 4. Expressiveness aspects Chou's approach

4.3.1.2 Conformity to the UML Standard

The language does not extend UML to define a new language for process modeling. Rather, it proposes to reuse UML1.4 activity and class diagrams. These diagrams represent the high-level part of the language. At the lower level, the author uses a proprietary object-oriented language for representing the process as a program.

4.3.1.3 Graphical representations and support of Multiple Views

The author proposes the use of two diagrams called *P-activity* diagram and *P-class* diagram. These diagrams are respectively based on a subset of the UML1.4 *Activity* and *Class* diagrams [70]. The P-activity diagram is used to model activities, their sequencing and synchronization, events and exception handlers. The P-class diagram is used to model products, roles, tools, schedules, budgets and their relationships. All these elements are in fact represented as UML classes with attributes and operations. However, there is no link between the two diagrams. They are just used for the process comprehension.

Example of a process representation using Chou's approach

As in Di Nitto's approach, Chou uses class diagrams to model process constituents and activity diagrams to model the flow of work. Thus, the representation of the process example will be the same as the one given in figure 3, section 4.2.1.2. At the lower level, the process modeler has to manually write the process program using Chou's OO process language.

4.3.1.4 Executability

While the P-class and P-activity diagrams are provided as a means to reason about the process, the approach does not provide an automatic generation of process programs from these diagrams towards the proprietary process execution language defined by the author. The process program has to be implemented by developers according to what is defined within the diagrams.

4.3.1.5 Modularity

Modularity is not addressed by the author.

4.3.1.6 Formality

No approach for process model validation is proposed. UML diagrams used for modeling processes and process programs used for process enactment are completely independent one from one another.

4.3.1.7 Tooling Support

No prototype is provided by the author.

Discussion

In Chou's approach, process constituents are represented as instances of the UML *Class* metaclass, which might not fit the semantics of software process constituents. One drawback of this approach is the lack of automatic generation of process programs from P-x diagrams, which requires a complete rewriting of the process by developers mastering the proprietary OO language the author proposes. Any addition to the P-class diagram imposes the coding of a new class and most of all, its linking with the other process classes.

4.4 Combining meta-modeling and executability

The UML4SPM language was developed in the context of the ModelWare [71] and ModelPlex [72] European projects. It allows the definition of process models which can be simulated and executed straightforwardly and without any transformation step thanks to the *Execution Model approach* [50] [73].

4.4.1 UML4SPM Evaluation

4.5.1.1 Semantic Richness

Table 5.summarizes expressiveness aspects of UML4SPM:

Approach	UML4SPM
Semantic Richness	
Core Process Elements	
Activity	Software Activity
Role	Responsible Role
Artifact	WorkProduct
Agent	Agent and Team
Tool	Tool
Activities Coordination	
Proactive Control	Ensured thanks to the combination of control flow, object flow, Invocation Actions and control nodes. Only the *start-finish* precedence relation is lacking.
Reactive Control	Ensured through events (*Message, Change* and *Time events*) and actions (*AcceptEvent* and *SendSignal* actions)
Exception Handling	Use of *RaiseExceptionAction* and *ExceptionHandler* concepts

Table 5. Expressiveness aspects UML4SPM

4.4.1.2 Conformity to the UML Standard

UML4SPM comes in form of a MOF2.0-compliant metamodel. It contains two packages: (1) the *UML4SPM Process Structure* package, which contains the set of primary process elements; and (2) the *UML4SPM Foundation* package, which contains the subset of UML2.0 concepts extended by these process elements.

4.4.1.3 Graphical representations and support of Multiple Views

In UML4SPM, *Activity* diagrams are used to model the sequencing of *Software Activities* and *WorkProducts* ex-

change between *Actions*. It is considered as the principal diagram and describes the process flow of work and the different roles involved in the process. This diagram is used for comprehension and training purposes. It is also the source of the process execution support.

Example of a process representation using UML4SPM

In order to be more intuitive, the UML2.0 Activity notation was slightly enriched in order to take into account some features proper to software process modeling. Main additions - not all presented in the diagram, figure 4- concern the ability to express extra information within the activity notation, such as roles, pre and post conditions, priority of the activity, its complexity, whether it is human (H) or machine oriented (M), accepted and triggered events, exception handlers, and so on. It is important to note that this extension does not affect the comprehension of people familiar with the use of UML2.0 Activity diagrams. The difference with UML1.x activity diagrams is that in figure 4., the *"Elaborate Analysis Model"* is not an activity but a *CallBehaviorAction* that, thanks to the complete arrow symbol, synchronously calls the *"Elaborate Analysis Model"* activity defined in another diagram. The complete notation of UML4SPM is given in [74].

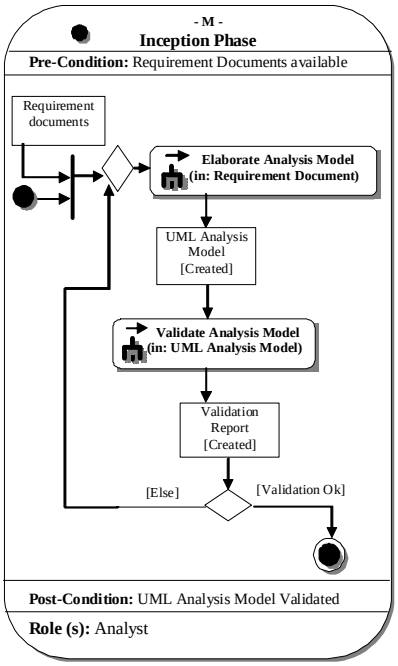


Figure 4. Representing a process example using UML4SPM

4.4.1.4 Executability

For executing UML4SPM process models, two approaches have been explored. The first one consists in reusing business process execution engines by mapping UML4SPM models towards BPEL [20]. Mapping rules and a prototype were defined in [63]. The second approach consisted in developing an operational semantics with an explicit *Execution Model* for UML4SPM. Each element of the UML4SPM metamodel is provided with an *eval* function that directly describes its effect on the *Execution Model*. Then, a UML4SPM process model can be executed straightforwardly without any transformation steps. The only condition is that process models are well-formed. The authors provide a Java implementation of the *Execution Model*. More details on the UML4SPM *Execution Model* can be found in [74].

4.4.1.5 Modularity

To compose a new *Software Activity* from other *Software Activities* one can take advantage of the flexibility offered by the UML2.0 *CallBehaviorAction*. The *CallBehaviorAction* allows two *Activities* to be interconnected in a practical way. The advantage of this construct is that behaviors are invoked as methods in classical programming languages. This way, modellers do not have to carry out the *unification* of *Software Activities* inputs and outputs (i.e., making their names identical). *CallBehaviorAction* being a *CallAction,* casting of parameters is done implicitly when activities are invoked thanks to the abstraction given by the UML *InputPins* and *OutputPins* concepts.

4.4.1.6 Formality

UML4SPM mainly reuses UML2.0 activity diagrams as a basis for software process modeling. A definition of the formal semantics of UML2.0 Activity diagrams was provided by many works in the literature [26], [27] and [28]. Most approaches consisted in mapping activity diagram concepts into Petri nets in order to perform model analysis and verification. These techniques have already been used to verify SPEM2.0 process models, so they could be reused for UML4SPM [65].

If we consider that the result of transforming UML4SPM models into BPEL is the actual definition of the process, then verification needs to be applied on BPEL process descriptions. Like for activity diagrams, many efforts have been made in order to define a formal semantics to BPEL. Most of them use the ASM formalism (Abstract State Machine), which seems to be the more appropriate for achieving correctness and completeness of the dynamic behaviours of the language [75] [76]. Others rely on Petri nets for static analysis of BPEL descriptions [77] [78].

4.4.1.7 Tooling Support

Authors propose a UML4SPM editor and an interpreter, both integrated into the Eclipse development environment. The editor is automatically generated from the UML4SPM metamodel, which adds a degree of flexibility in case of the evolution of the language. The interpreter comes in form of Java classes representing the UML4SPM *Execution Model*. UML4SPM process models once edited can straightforwardly be executed within the same environment and it is completely transparent to the process modeler. The prototype can be downloaded at [74].

Discussion

For people who are not familiar with the newly defined UML2.0 standard, it may take quite a long time to get familiar with the use of UML2.0 control nodes, events, pins, actions and so on. This can then hamper the use of the UML4SPM language. However, UML4SPM proposes to deal with one kind of diagram (i.e. Activity diagrams) for modeling and executing software processes. The approach proposes two ways of supporting process enactment and simulation. The first one relies on a mapping into BPEL while the second one enables the production of

executable software process models through the *Execution Model* approach. Finally, *modularity* is addressed thanks to *CallBehaviorActions*. This is fully automated when auxiliary software activities (i.e., activities with their *isInitial* property set to false) are added to a main process (i.e. a software activity having its *isInitial* property set to true). However, in case of composing different processes having all their *isInitial* property set to true, one of them have to be designated as the main process and the *isInitial* property of the remaining ones have to be set to false.

5 Comparative Evaluation

The aim of this section is to discuss the merits and limits of each of the studied approaches and to answer some of the questions raised in the introduction of this paper.

Which approach best fits my needs?

In the previous sections, we evaluated UML-based approaches for modeling software processes. Each one addressing different process aspects, it is not possible to nominate the *universally best* approach. However, herein we summarize some of the observations we made while comparing the different systems according to the PML requirements. This may help in choosing the most suitable approach with respect to project and team specific needs.

To give a summary picture of this comparison, we propose to rate approaches with respect to PML requirements with the following values: "0" when a requirement is not supported by the approach, "1" when it is partially supported, and "2" when it is fully. For composite requirements such as "*semantic richness"*, the same notation is applied for each of its components, and combined as follows: "0"if all the components of the composite requirement are rated at "0", "2" if all the components of the composite requirement are rated at "2", and "1" otherwise.

While being quite partial and subjective, this evaluation makes it easier for a decider to identify the language answering her expectations. Of course, the reader still needs to refer to the detailed review given in this paper and to the original work of approach's authors in order to evaluate more precisely the potential of each system. This is particularly true when two or more approaches have the same rate for an atomic requirement (i.e. not composed of other requirements). As an example, all the approaches have the rate 2 for the "*Conformity to the standard*" and "*Graphical representations and support of multiple views*" requirements. In case where all the approaches have the same rate for a composite requirement, then the reader is encouraged to look at the rate of the components of this requirement in order to have a more precise evaluation.

The result of evaluating the reviewed approaches is given in table 6. In case where an approach obtains the highest rate regarding a requirement (i.e. 2), this is highlighted by making the rate **bold**. Hereunder, for each requirement, we discuss the contribution of each approach and we identify the most adequate approaches with respect to it.

Semantic richness

Most approaches feature all common process elements except the notion of *Agent*, which is lacking in three approaches out of six (SPEM1.1, SPEM2.0 and Chou's approaches). *Exception handling* is also missing in all the approaches except Chou's and UML4SPM ones. *Proactive control* is more widely supported than *Reactive contro*l, which in most cases is provided by means of events modeled within state diagrams. SPEM1.1 & SPEM2.0 do not feature *Reactive Control* at all. Table 6 gives a summary of the semantic richness capabilities of each approach. According to this summary, UML4SPM followed by Di Nitto's and Chou's approaches are those that provide more capabilities regarding the expressiveness requirement while SPEM1.1 is the less expressive one.

Executability

Regarding executability, Chou's approach consists in manually rewriting the process program from the UML diagrams in order to execute the process. Di Nitto's approach consists in generating code from the three UML1.3 diagrams used for describing the process (i.e., *Activity*, *Class* and *State Machine*). However, no information is given about how process aspects (i.e., activity sequencing, events, actions, class's operations and attributes) defined in these diagrams are translated and integrated into the Java code. UML4SPM proposes two approaches for executing process models. The first one promotes the reuse of existing BPEL process engines, but it requires a configuration phase. The second one makes it possible to execute and to simulate process models straightforwardly, without any transformation of configuration step, based on the notion of *Execution Model*. However, the current UML4SPM process engine still does not integrate all the utilities related to resources allocations and management. Looking at the summary table, we can see that Di Nitto's approach and UML4SPM are the only approaches that provide an automatic execution support for process models while Chou's approach still requires hand-coding the process in order to execute it. SPEM1.1, SPEM2.0 and Promenade do not offer any execution support.

Conformity to the UML standard

Three of the six approaches we discussed (i.e., Di Nitto et al., Promenade and Chou approaches) do not define new concepts nor extend UML metamodel ones. They simply provide a predefined UML class diagram for defining main process constituents (i.e., *Activity, Role, Artifact, Agent or Tool*) in terms of instances of the UML1.x *Class* metaclass (i.e., simple UML classes). Thus, these process elements do not have specific semantics and notations: they borrow them from the UML *Class* metaclass. SPEM1.1, SPEM2.0 and UML4SPM are the only SPMLs that provide a set of concepts with their own semantics and notations through metamodel and profiling mechanisms instead of simply using UML diagrams. SPEM2.0 redefines many concepts from scratch instead of simply reusing them from the UML2.0 activity and action packages. SPEM1.1 makes quite a good trade-off. It offers simple concepts through a MOF metamodel extending UML1.4 concepts. Still, all the approaches conform to the UML standard in one way or another, which justifies the

rate 2 for all the systems in the evaluation table.

Graphical Representations and Support of Multiple Views

The activity diagram remains the most used diagram in all the approaches for representing the process workflow except in SPEM2.0. The Di Nitto, SPEM1.1, Chou and Promenade approaches use extensively the class diagram in order to represent basic process elements and their relations. Additionally, Di Nitto's approach uses state machines in order to represent process element's lifecycles. SPEM2.0 does not rely on UML diagrams for representing processes but provides a behavioral diagram based on a set of proprietary notations. UML4SPM uses mainly the activity diagram

Modularity

SPEM2.0 provides a range of mechanisms for extending and combining chunks of process and method descriptions, which justifies its highest rate in the evaluation table. SPEM1.1 approach for composing process models uses on a name-based unification procedure (i.e. renaming process elements) and presents some limitations as discussed in section 4.1.1.4. UML4SPM is also weak in offering a completely automated solution for composing different processes. Modularity is not supported by the remaining approaches.

Formality

Formality is the main weakness of most of the approaches. We saw that most of them do not provide a built-in formal semantics. Still, model-checking techniques can be used in order to verify some properties of the process. This is can be done for instance in the case of SPEM1.1, SPEM2.0 and UML4SPM by translating process descriptions into Petri nets (the semantics then being in the translation). In Di Nitto's, Promenade, and Chou's approaches, the process description is scattered in different and independent diagrams, which makes it difficult to ensure process consistency.

Tooling Support

Finally, regarding the *Tooling Support*, only the industrial standards (i.e., SPEM1.1 and SPEM2.0) have some well-supported implementations. This justifies the rate 2 in the evaluation table. However, the standard implementation differs among tools and process models are stored in proprietary formats, making process model exchanges quite impossible. In Di Nitto's, Chou's and UML4SPM approaches, prototypes do exist but we had access to the UML4SPM process engines only. No tool or prototype does exist for Promenade.

6 CONCLUSION

This paper provided an extensive overview of predominant UML-based languages for software process modeling. We have framed a set of requirements for process modeling language designs. This set of requirements has been used as a basis for the evaluation and the comparison between the various approaches. Beyond providing a detailed comparison, we also evaluated the suitability of UML as a process modeling language and we highlighted its advantages as well as its drawbacks. We showed that whilst UML offers a high potential regarding understandability – using graphical representations and expressiveness, it still fails in offering executable and formal process models. We also saw how the studied approaches reuse UML as a building block for modeling software processes and how each of them tried to overcome UML limits by offering its own solutions. Each approach providing a different set of capabilities and ways to address process modeling issues (e.g., standardization, reuse, execution, expressiveness, etc.), it remains up to project managers, according to the result of this evaluation, to choose the approach that best fits their expectations.

Approaches / Requirements	SPEM1.1	SPEM2.0	Di Nitto's et al.	Promenade	Chou's Approach	UML4SPM
Semantic Richness	**1**	**1**	**1**	**1**	**1**	**1**
Process Elements	1	1	1	2	1	2
Proactive Control	1	2	1	1	1	1
Reactive Control	0	0	2	1	2	2
Exception Handling	0	0	1	0	2	2
Conformity to UML	**2**	**2**	**2**	**2**	**2**	**2**
Graphical Representations and Support of Multiple Views	**2**	1	**2**	**2**	**2**	1
Executability	0	0	**2**	0	1	**2**
Modularity	1	**2**	0	1	0	1
Formality	1	1	0	0	0	1
Tooling Support	**2**	**2**	1	0	1	1

Table 6: Summary of the evaluation of UML-Based approaches with regard to PML Requirements

REFERENCES

[1] S. M. Sutton Jr., P. L. Tarr, and L. J. Osterweil, *An Analysis of Process Languages, CMPSCI Technical Report 95-78*, University of Massachusetts, 1995.

[2] C. Montangero, J. C. Derniame, B. A. Kaba *et al.*, "The software process: Modelling and technology," *Lecture Notes In Computer Science;* , vol. 1500, pp. 1-14, 1999.

[3] ANSI/IEEE, *IEEE Standard for Software Verification and Validation Plans*, The Institute of Electrical and Electronics Engineers, Inc., 1987.

[4] S. M. Sutton Jr., D. Heimbigner, and L. J. Osterweil, "APPL/A: a language for software process programming," *ACM Trans. Softw. Eng. Methodol.*, vol. 4, no. 3, pp. 221-286, 1995.

[5] J. Lonchamp, "A structured conceptual and terminological framework for software process engineering," in Proceedings of the 2nd International Conference on Software Process, Los Alamitos, CA, 1993, pp. 41-53.

[6] F. Ruiz-Gonzalez, and G. Canfora, " Software Process: Characteristics, Technology and Environments " *UPGrade, The European Journal for the Informatics Professional*, vol. 5, pp. 6-10, 2004.

[7] G. E. Kaiser, N. S. Barghouti, and M. H. Sokolsky, "Preliminary experience with process modeling in the MARVEL software development environment kernel," in Proceedings of the 23rd Annual Hawaii International Conference on System Sciences, Hawaii, USA, 1990, pp. 131-140.

[8] S. Bandinelli, A. Fuggetta, and C. Ghezzi, "Software Process Model Evolution in the SPADE Environment," *IEEE Trans. Soft. Eng.*, vol. 19, no. 12, pp. 1128-1144, 1993.

[9] R. Conradi, M. L. Jaccheri, C. Mazzi *et al.*, "Design, Use and Implementation of SPELL, a language for Software Process Modelling and Evolution," in Proceedings of the 2nd European Workshop on Software Process Technology, 1992, pp. 167-177.

[10] E. Di Nitto, L. Lavazza, M. Schiavoni *et al.*, "Deriving executable process descriptions from UML," in Proceedings of the 24th International Conference on Software Engineering, Orlando, Florida, 2002, pp. 155-165.

[11] B. Henderson-Sellers, and C. A. González-Pérez, "A Comparison of Four Process Metamodels and the Creation of a New Generic Standard," *Information and Software Technology*, vol. 47, no. 1, pp. 49-65, 2005.

[12] A. Finkelstein, J. Kramer, and B. Nuseibeh, *Software process modelling and technology*: Advanced Software development Series, John Wiley & Sons Inc, 1994.

[13] OMG, *"Workflow Management Facility Specification v1.2", OMG document formal/00-05-02, at* http://www.omg.org/spec/WfMF/, 2000.

[14] OMG, *SPEM1.1, "Software Process Engineering Metamodel", OMG document formal/05-01-06, at* http://www.omg.org/cgi-bin/doc?formal/05-01-06/, 2005.

[15] M. Dumas, and A. ter Hofstede, H. M. , "UML Activity Diagrams as a Workflow Specification Language," in Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 2001, pp. 76-90.

[16] W. Van der Aalst, A. ter Hofstede, B. Kiepuszewski *et al.*, " Workflow patterns," *Journal of Distributed and Parallel Databases*, vol. 14, no. 3, pp. 5–51, 2003.

[17] S. White, "Process modeling notations and workflow patterns," *Workflow Handbook 2004*, L. Fischer, ed., pp. 265–294, FL, USA: Future Strategies Inc., Lighthouse Point, 2004.

[18] S. Jablonski, and C. Bussler, *Workflow Management: Modeling Concepts, Architecture and Implementation*, London,UK.: Thomson Computer Press, 1996.

[19] P. Wohed, W. van der Aalst, M. Dumas *et al.*, "Pattern-based analysis of UML activity diagrams," in Proceedings of the 25th International Conference on Conceptual Modeling (ER'2005), Klagenfurt, Austria, 2005, pp. 63-78.

[20] OASIS, *Web Services Business Process Execution Language Version 2.0. Working Draft. WS-BPEL TC* OASIS, January 2007.

[21] N. Russell, A. ter Hofstede, D. Edmond *et al.*, "Workflow data patterns: Identification, representation and tool support," in Proceedings of the 25th International Conferenceon Conceptual Modeling, Klagenfurt, Austria, 2005, pp. 353-368.

[22] N. Russell, W. van der Aalst, A. ter Hofstede *et al.*, "Workflow resource patterns:Identification, representation and tool support," in Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05), Porto, Portugal, 2005, pp. 216–232.

[23] OMG, *Semantics of a Foundational Subset for Executable UML Models RFP, OMG document ad/05-04-02, at:* http://www.omg.org/spec/FUML/1.0/Beta1/, 2005.

[24] V. Vitolins, and A. Kalnins, "Semantics of UML 2.0 Activity Diagram for Business Modeling by Means of Virtual Machine," in Proceedings of the 9th IEEE International EDOC Enterprise Computing Conference, 2005, pp. 181-194.

[25] STL. "STL: The UML2,0 Semantics Project," http://www.cs.queensu.ca/~stl/internal/uml2/.

[26] H. Störrle, and J. H. Hausmann, " Towards a Formal Semantics of UML 2.0 Activities," in Proceedings of Liggesmeyer, K. Pohl, and M. Goedicke, editors, Software Engineering, 2005, pp. 117-128.

[27] J. P. Barros, and L. Gomes, "Actions as Activities and Activities as Petri nets," in Workshop on Critical Systems Development with UML, Seiten, 2003, pp. 129-135.

[28] S. Sarstedt, "Semantics Foundation and Tool Support for Model-Driven Development with UML2 Activity Diagrams, PhD document," Ulm University, 2006.

[29] B. Curtis, M. I. Kellner, and J. Over, "Process modeling," *Communications of the ACM*, vol. 35, no. 9, pp. 75-90, 1992.

[30] M. L. Jaccheri, M. Baldi, and M. Divitini, " Evaluating the Requirements for Software Process Modelling Languages and Systems" in Process support for Distributed Team-based Software Development (PDTSD'99), Florida, USA, 1999, pp. 570-578.

[31] K. Z. Zamli, and P. A. Lee, "Taxonomy of process modeling languages," *ACS/IEEE International Conference on Computer Systems and Applications*, pp. 435-437, 2001.

[32] P. Armenise, S. Bandinelli, C. Ghezzi *et al.*, "A Survey and Assessment of Software Process Representation Formalisms," *Int. Journal on Software Engineering and Knowledge Engineering*, vol. 3, no. 3, pp. 401-426, 1993.

[33] C. Schlenoff, A. Knutilla, and S. Ray, "Unified Process Specification Language: Requirements for Modeling Process," *Interagency Report*, vol. 5910, 1996.

[34] M. Dowson, B. Nejmeh, and W. Riddle, "Fundamental Software Process Concepts" in 1st European Workshop on Software Process Modeling, Milan, Italy, 1991, pp. 15-37.

[35] R. Conradi, C. Fernstr, A. Fuggetta *et al.*, "Towards a Reference Framework for Process Concepts," in 2nd European Workshop on Software Process Technology, 1992, pp. 3-17.

[36] P. H. Feiler, and W. S. Humphrey, "Software process development and enactment: concepts and definitions," in 2nd International Conference on Software Process. 'Continuous Software Process Improvement' Berlin, Germany, 1993, pp. 28-40.

[37] J. Lonchamp, "A structured conceptual and terminological framework for software process engineering." pp. 41-53.

[38] A. Fuggetta, "Software process: a roadmap," in Conference on The Future of Software Engineering, Limerick, Ireland, 2000, pp. 25-34.

[39] A. Wise, B. S. Lemer, E. McCall, K. *et al.*, "Using Little-JIL to Coordinate Agents in Software Engineering," in 15th IEEE international conference on Automated Software Engineering, 2000, pp. 155-165.

[40] C. Popescu, and C. Charoenngam, *Project Planning, Scheduling, and Control in Construction: An Encyclopedia of Terms and Applications*: Wiley-Interscience, 1995.

[41] M. Klein, and C. Dellarocas, "A Knowledge-based Approach to Handling Exceptions in Workflow Systems," *Computer Supported Cooperative Work (CSCW)*, vol. 9, no. 3, pp. 399-412, 2000.

[42] L. Briand, Y. Labiche, M. Di Penta *et al.*, "An Experimental Investigation of Formality in UML-Based Development," *IEEE Trans. Softw. Eng.*, vol. 31, no. 10, pp. 833-849, 2005.

[43] W. McUmber, E. , and B. Cheng, H. C., "A general framework for formalizing UML with formal languages," in Proceedings of the 23rd International Conference on Software Engineering, Toronto, Ontario, Canada, 2001, pp. 433-442.

[44] A. S. Evans, and S. Kent, "Meta-modelling semantics of UML: the pUML approach," in 2nd International Conference on the Unified Modeling Language, Colorado, USA, 1999, pp. 140-155.

[45] X. Franch, P. Botella, X. Burgus *et al.*, "ComProLab: A Component Programming Laboratory," in Proceedings 9th Software Engineering and Knowledge Engineering Conference, 1997, pp. 397-406.

[46] X. Franch, and J. Rib, "A Structured Approach to Software Process Modelling," in Proceedings of the 24th Conference on EUROMICRO - Volume 2, 1998, pp. 753-762.

[47] S.-C. Chou, “A process modeling language consisting of high level UML diagrams and low level process language,” *Journal of Object-Oriented Programming*, vol. 1, no. 4, pp. 137-163, 2002.
[48] OMG, *SPEM1.0, “Software Process Engineering Metamodel”, OMG document formal/02-11/14, at* http://www.omg.org/cgi-bin/doc?formal/02-11-14/, 2002.
[49] OMG, *UML1.4, "Unified Modelling Language", version 1.4., OMG document formal/01-09-67, at* http://www.omg.org/spec/UML/1.4/, 2001.
[50] R. Bendraou, M.-P. Gervais, and X. Blanc, “UML4SPM: A UML2.0-Based metamodel for Software Process Modeling,” in Proceedings of ACM/IEEE 8th MoDELS/UML, Montego Bay, Jamaica, 2005, pp. 17-38.
[51] B. Combemale, A. Caplain, X. Cregut *et al.*, “Towards a rigorous process modeling with SPEM,” in Proceedings of the ICEIS'06, Paphos, Cyprus, 2006.
[52] IBM. "Rational Process Workbench (RPW)," http://www-128.ibm.com/developerworks/rational/library/6001.html#author.
[53] Osellus. "Osellus IRIS Suite," www.Osellus.com.
[54] Softeam. "Objecteering," http://www.Objecteering.com.
[55] OMG, *SPEM2.0, "Software Process Engineering Metamodel", OMG document, final adopted specification, ptc/07-03-03, at* http://www.omg.org/spec/SPEM/2.0/, 2007.
[56] OMG, *UML2.1.1, "Unified Modeling Language", Infrastructure Specification, version 2.1.1., OMG document formal/07-02-06 ,at* http://www.omg.org/spec/UML/2.1.1/, 2007.
[57] OMG, *Diagram Interchange, adopted specification, OMG document formal/06-04-04, at* http://www.omg.org/spec/UMLDI/, 2006.
[58] OMG, *UML2.1.1, "Unified Modeling Language", Superstructure Specification, version 2.1.1., OMG document formal/07-02-04 , at* http://www.omg.org/spec/UML/2.1.1/, 2007.
[59] OMG, *BPMN, Business Process Modeling Notation final adopted specification, OMG document dtc/06-02-01, at* http://www.omg.org/spec/BPMN/1.2/, 2006.
[60] OMG, *SPEM2.0 RFP, “Software Process Engineering Metamodel”, OMG document ad/2004-11-04, 2004, at* http://www.omg.org/docs/ad/04-11-04.pdf, *page last visit October 8, 2009*, 2004.
[61] IBM. "Rational Portofolio Manager (RPM)," http://www-306.ibm.com/software/awdtools/portfolio/index.html.
[62] MPM. "Microsoft Project Manager "; http://www.microsoft.com/france/office/2007/solutions/epm/overview.msp.
[63] R. Bendraou, A. Sadovykh, M.-P. Gervais *et al.*, “Software Process Modeling and Execution: The UML4SPM to WS-BPEL Approach,” in Proceedings of 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, 2007, pp. 314-321.
[64] C. A. Gunter, and D. S. Scott, *Semantic domains*: MIT Press, Cambridge, MA, USA, 1991.
[65] R. Bendraou, B. Combemale, X. Cregut *et al.*, “Definition of an Executable SPEM 2.0,” in Proceedings of the 14th Asia-Pacific Software Engineering Conference, 2007, pp. 390-397.
[66] EPF. "Eclipse Process Framework (EPF)," www.eclipse.org/epf/.
[67] IBM. "IBM Rational Method Composer (RMC)," www.ibm.com/software/awdtools/rmc/
[68] OMG, *UML1.3, "Unified Modelling Language", version 1.3., OMG document formal/00-03-01, at* http://www.omg.org/spec/UML/1.3/, 2000.
[69] G. Cugola, E. Di Nitto, and A. Fuggetta, “The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS,” *IEEE Trans. Softw. Eng.*, vol. 27, no. 9, pp. 827-850, 2001.
[70] S.-C. Chou, “Process Program Development Based on UML and Action Cases, Part 1: the Model,” *Journal of Object-Oriented Programming*, vol. Vol. 13, no. 2, pp. 21-27, 2000.
[71] Modelware. "Modelware, IST European Project contract no 511731," http://www.modelware-ist.org/.
[72] Modelplex. "Modelplex, IST European Project contract IST-3408," http://www.modelplex-ist.org/.
[73] R. Bendraou, M.-P. Gervais, and X. Blanc, “UML4SPM: An Executable Software Process Modeling Language Providing High-Level Abstractions,” in Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference, 2006, pp. 297-306.
[74] R. Bendraou. "UML4SPM Publications, Prototype, notation and evaluation using the ISPW-6 Process example," http://pagesperso-systeme.lip6.fr/Reda.Bendraou/Publications.htm.
[75] R. Farahbod, U. GlÄasser, and M. Vajihollahi, “Specification and Validation of the Business Process Execution Language for Web Services,” *In Abstract State Machines*, pp. 78-94, 2004.
[76] D. Fahland, and W. Reisig, “ASM-based semantics for BPEL: The negative control flow,” in In Proceedings of the 12th International Workshop on Abstract State Machines, Paris, France, 2005, pp. 131-151.
[77] S. Hinz, K. Schmidt, and C. Stahl, “Transforming BPEL to Petri nets” in in Proceedings of 3rd International Conference on Business Process Management, 2005, pp220-235.
[78] C. Ouyang, W. M. P. van der Aalst, S. Breutel *et al.*, *Formal Semantics and Analysis of Control Flow in WS-BPEL* BPM Center Report BPM-05-13, BPMcenter.org, 2005.