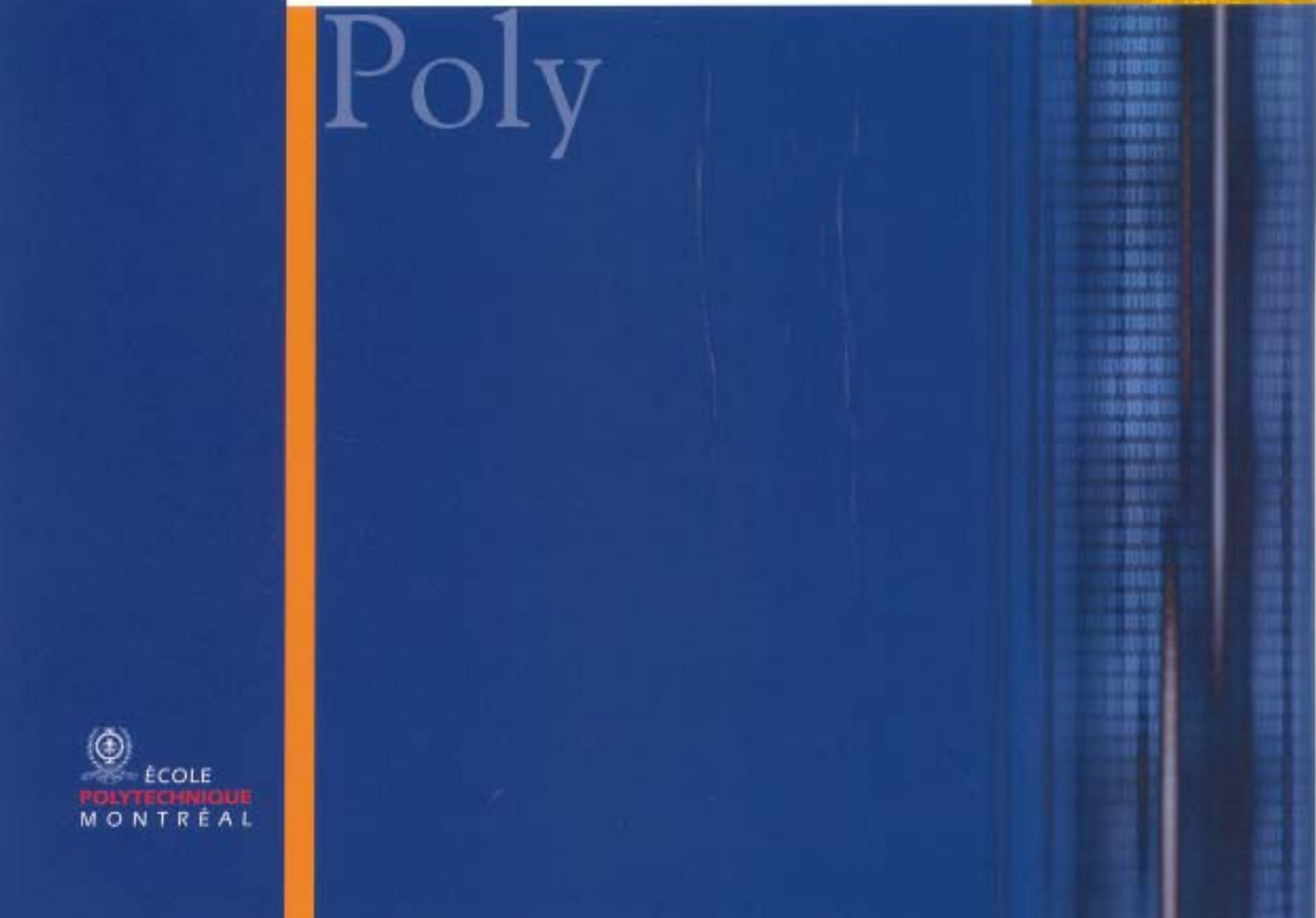
EPM–RT–2010-15

DSL4SPM : DOMAIN-SPECIFIC LANGUAGE FOR SOFTWARE PROCESS MODELING

Noureddine Kerzazi, Pierre N. Robillard
Département de Génie informatique et génie logiciel
École Polytechnique de Montréal

Novembre 2010



EPM-RT-2010-15

DSL4SPM: Domain-Specific Language for Software Process Modeling

Noureddine Kerzazi, Pierre N. Robillard
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

Novembre 2010

Dépôt légal :
Bibliothèque nationale du Québec, 2010
Bibliothèque nationale du Canada, 2010

EPM-RT-2010-15
DSL4SPM: Domain-Specific Language for Software Process Modeling
par : Noureddine Kerzazi, Pierre N. Robillard
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

DSL4SPM: Domain-Specific Language for Software Process Modeling

Noureddine Kerzazi, Pierre N. Robillard
Department of Computer and Software Engineering
École Polytechnique de Montréal
Montréal, Canada
{noureddine.kerzazi, pierre-n.robillard}@polymtl.ca

Abstract – This paper presents a novel formal approach to software process modeling based on the Software Process Engineering Metamodel (SPEM) for the syntactic aspect of a process model and a domain-specific language (DSL) for the semantic aspect of the model. This approach provides a conceptual framework for designing processes in a more abstract way and to enable process implementation on various platforms. A Process-Centered Software Environment (PCSE) called DSL4SPM (Domain-Specific Language for Software Process Modeling) is a plug-in tool which satisfies the meta-requirements for Process Modeling Languages (PMLs). The key concept of the DSL4SPM is the use of a toolbox, containing SPEM elements, to instantiate objects in a graphical scene. The process model designer links these elements with relations, and defines the values of the attributes required for both these and the objects, with the aim of arriving at a consolidated view of the problem. An overview of the advantages of the approach is presented. With it, the process manager is able to quickly and easily model a process from innovative perspectives, with the aim of better understanding the risks associated with software development.

Keywords: Process-Modeling Languages (PML); Process-Centered Software Environment (PCSE); Software Process; Software Process Engineering Metamodel (SPEM); Domain-Specific Language (DSL); Domain-Specific Language for Software Process Modeling (DSL4SPM).

1. Introduction

The lack of a conceptual framework for software process modeling limits perception of the risks associated with software development activities. The use of the tool EPF Composer[1] in previous work [2] revealed the solution to the issue underlying the textual description of processes, which is to define all the concepts needed to structure a process before they are used (static instantiation [3]). Furthermore, UML notation even when extended by specific profiles, is still incomplete for process modeling and needs to be supported by an operational semantics.

The Model-Driven Architecture (MDA) paradigm [4] introduces the concept of the platform-independent model. This involves modeling the solution in an abstract way and in the language of the application domain, independently of the platform on which the process will be implemented.

For the concept to be valid at the metamodel level, a process engineer must address several concerns with respect to a process model (e.g. activity, risk, knowledge, measurement) to generate, by transforming that model, a result suitable for a particular platform (a website, for example).

This project presents a new formal approach to Model-Driven Process (MDP) modeling. The central idea is that the use of models can offer a conceptual framework for defining processes in a more abstract way, with the aim of implementing them on various platforms (Visual Studio Team System (VSTS), websites, or a project planning system, like MS Project or IBM's Rational Portfolio Manager). This approach is based on the Software Process Engineering Metamodel (SPEM)[5] for the syntactic aspect of the process model and on a domain-specific language (DSL) [6] for the semantic aspect, providing the structural and behavioral views of the problem respectively.

The DSL4SPM tool was developed to demonstrate the potential of such a conceptual approach to process modeling. To illustrate the benefits of this tool and to explore the advantages of the proposed MDP approach, we modeled a maintenance process in conformity with the ISO/IEC 14764:2006 standard [7].

This paper is organized as follows: Section 2 outlines the evolution of the process modeling domain and related work. Section 3 gives a conceptual view of the formal approach that we are proposing. Section 4 describes the DSL4SPM tool and illustrates some of its important functionalities. Section 5 presents concluding remarks.

2. Related Work

This section presents a synthesis view of the evolution of Process-Modeling Languages (PMLs) and PCSEs. PMLs are languages, defined in a formal or semi-formal way, designed to clarify software development activities. Their aim is to define, analyze, and improve a methodology (software process improvement: SPI). Process-Centered Software Environments (PCSEs, also called PCEs or PSEEs) are software systems which support the modeling, automation, and instantiation of a software development process. The goal of this synthesis is to analyze the relationship between the technological paradigm of software process and the technological paradigm adopted for PCSEs. Figure 1 shows the most relevant technological steps in the evolution of PMLs and their corresponding PCSEs.

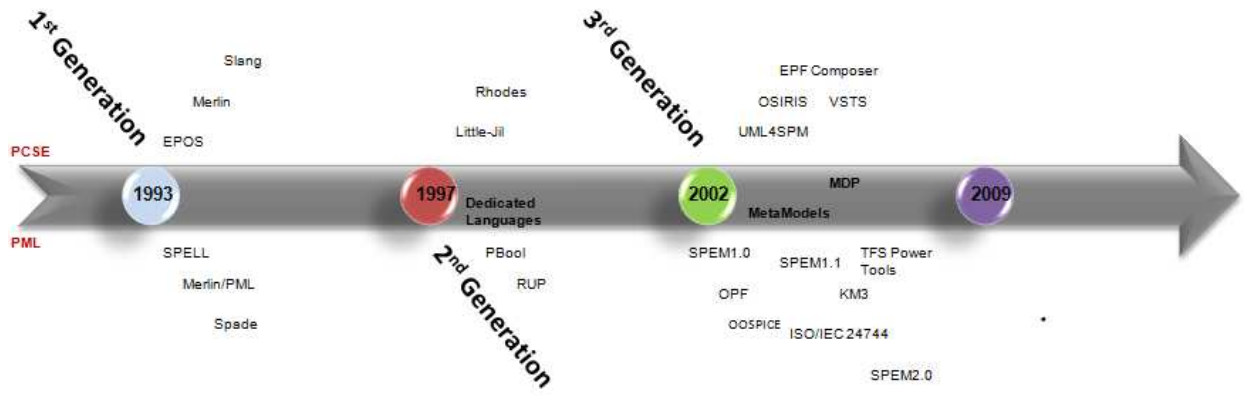


Fig. 1. Evolution of MPLs and PCSEs

First-generation PCSEs were derived from the use of generic computer programming languages [8]. This approach, characterized as declarative and functional, does not permit the dynamic and incremental evolution of process models, as it is too restrictive to support flexibility in the structuring of the model components (e.g. [9-13]).

Second-generation systems [8] are based on more specialized languages (e.g. [14-18]) or on graphs and transitional networks. Several computer programming languages emerged at this time as well. PCSEs of this generation focus on the scheduling of activities from a dynamic point of view, but do not provide an organizational view of the process model.

Third-generation systems are known for the definition of the OMG standard: the Software Process Engineering Metamodel (SPEM) [5], which was designed to harmonize PML definitions and concepts. The harmonization approach is based on the use of UML for PMLs and the Object Oriented (OO) paradigm for PCSEs. Languages such as UML and OO have gained broad acceptance in the software industry. SPEM provides a static view of the process centered on the Role-Activity-Artifact elements. However, it does not offer a dynamic representation of the behavioral aspects of the process, the importance of which was underlined by Curtis [15] when the first-generation systems emerged. The OMG suggests the use of UML (e.g. chart of activity and state) to fill this gap. Although based on UML, PCSEs of this generation are oriented towards the creation of static content aimed at defining the concepts and practices that describe software development activities – the project's life cycle (e.g. IBM Rational Method Composer, Microsoft Solution Framework, and EPF Composer). They are also seen as having semantic weaknesses.

Following this evolution, four metamodels have been compared and a new model has been proposed , also based on UML, [19] which would become ISO/IEC 24744 [20], the basis and concepts of which are described in [21].

It is clear from this evolution that a concern for standardization had emerged in the use of metamodels by third-generation systems. These metamodels are based on UML and the OO paradigm. However, a new Model Driven Architecture (MDA) paradigm is becoming established. This paradigm is based on three principles:

- Direct representation: focusing on the problem domain concepts using a DSL, instead of attempting to adapt UML to the domain. The result is precise, semantically rich, and easy to modify.
- Automation: using transformation tools to map the model to the target platform, which bridges the semantic gap between the domain concepts and the language of implementation. So, from the same process model, the designer can generate a process in conformity with the VSTS, a website, or a project planning system like MS Project or IBM's Rational Portfolio Manager.
- Open standards: such as XML, on which DSLs can be based for improved interoperability.

The key to the MDA paradigm is the use of a DSL. Based on his experience with both MDA and DSL Tools [6], Cook is convinced that it would be more advantageous to create a DSL adapted to the domain problem than to use a UML profile. His conclusion has given rise to a new research direction [22-25] related to process level, which we call the Model-Driven Process, or MDP.

In our work here, we formalize the MDP framework. This formalism is based on: 1) the SPEM standard, which is used for building the syntactic structure, and so providing a standardized static structural view; and 2) a new DSL, which is used to formalize the semantic relationships between SPEM elements, and so providing a dynamic behavioral view. This formal approach allows process designers to create, as well as to represent, analyze (from several points of view), validate, instantiate, and easily modify a process model.

3. DSL4SPM Foundation: Concept Overview

To formalize the MDP approach, a syntax and one or more semantics (depending on the modeling perspectives adopted) are required. SPEM, which is widely accepted in industry and academia, is used for the syntax. To define a rich semantics, a DSL is created on the basis of a formalization of the relationships between the SPEM elements that form the structure of the process model. This

formalization creates views based on attributes to represent flows (activity, knowledge, risk, etc.). The conceptual details are presented in the following section.

3.1. Syntax of a process model

The SPEM 2.0 profile defines a set of stereotypes which extends UML 2. The framework for this model, which is illustrated in Figure 2, is composed of two containers labeled “Method Content” and “Process”.

In the “Method Content” container are the definitions of the basic elements: Role, Task, Work Product, and Guidance, which are used to build a process. All these elements must be defined in this container before being instantiated in a process. They are classified in “Categories” (e.g. Disciplines, Domains, Work Product Types, Role Set, Tools) to facilitate access to them. The Guidance element, which can take several forms, e.g. guidelines, checklists, examples, or templates, lies at the intersection of the “Method Content” and “Process” containers, because it can be used for either basic elements or instances of processes. The “Process” container can contain one or more instances of the Delivery Process, which are presented in a sequential list of activities describing the flow of work to be performed. “Activity” is a “Task Uses”, “Role Use”, and “Work Product Use” container. These latter elements are expressed on the basis of “Method Content” elements, in order to promote reusability. This reuse mechanism is also ensured by the “Capability Patterns” container, which serves as a container for reusable process blocks.

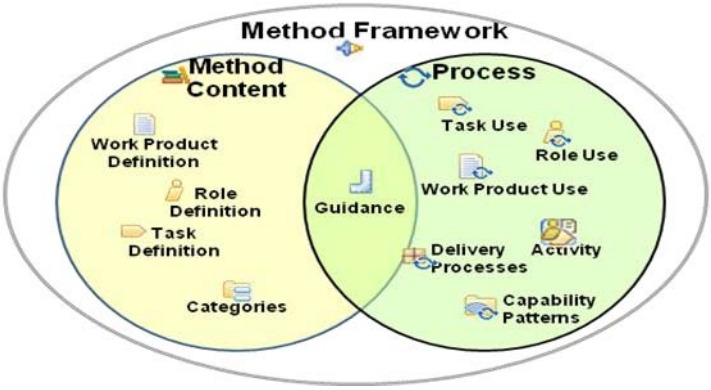


Fig. 2. SPEM Method Framework mapped to Method Content versus Process

It is important to note that SPEM defines a set of concepts with a graphical icon, and that a minimal set of attributes and associations is specified for each concept. However, this representation offers only a static view that refers to a list of activities. In addition to this representation, the process designer needs to address other views related to other concerns (e.g. knowledge, risk), which requires that another meaning be given to the same syntax, which is explained by a semantics.

In a model-based approach, a semantic is supported by tools that ensure the interpretation or transformation of the model to provide a precise direction. Essentially, such an interpretation involves a systematic mapping of expressions from one language to another, the semantics of which are well understood. For example, which attributes should we retain to represent the concept of risk or knowledge in a process model? To answer this question, we developed the DSL4SPM, which is a tool designed to interpret the properties of the relations between the SPEM elements with the aim of building other views to represent flows (e.g. knowledge, risk; see Figure 7 below on the generation of views starting with a reference set). These relations are formalized by ontological concepts.

3.2. Formalization of Relations: Semantics

The basic idea is that, with the same elements as those defined in "Method Content" with SPEM (left circle in Figure 2), it is possible to address other behavioral views using attributed relations. The concept of the attributed relation does not exist in UML, from which the SPEM profile was defined. This is one of the reasons why we chose the DSL paradigm.

Ontological view for formalizing relations

The semantic richness provided by our approach is based on the attributed relations between the SPEM elements. These relations are formalized by means of an ontological approach with a twofold objective: (1) to deduce a view based on the problem domain (see Figure 5 in the section on the description and characteristics of the tool); and (2) to validate the design of the model at the conceptual level using predetermined rules.

According to Gruber [26], an ontology is defined as the explicit specification of an abstract view of the problem domain. This specification describes the concepts of the domain and their properties, and the relations and constraints between them. Our domain is process modeling, and its concepts are those provided by SPEM and other metamodels which we have created to satisfy views of the domain other than the activities view. For each concept, the properties defined in the SPEM specification were categorized and enriched. In terms of the relationship formalism, we have adopted the axioms set out in [27].

An ontology is formally defined as a 2-tuple OntoSemantics : = ‹ $\boldsymbol{E}$, $\boldsymbol{R}$ ›, where

- $\boldsymbol{E}$ is a set of elements $E := \{e_1, e_2, \dots, e_n\}$;
- $\boldsymbol{R}$ is a set of relations $R := \{r_1, r_2, \dots, r_n\}$.

The following relations have been implemented, and examples of them are illustrated in the conceptual model in Figure 3.

a. Inverse Properties

An object property may have a corresponding *inverse* property. Given the property TaskHasSteps and its inverse property partOfTask, if Task $\mathbf{T_1}$ TaskHasSteps $\mathbf{Step_1}$, then the inverse property states that $\mathbf{Step_1}$ is partOfTask $\mathbf{T_1}$. For example, by inference, we can know the steps (Step) of each task, and we are enabled by the inverse property of the relationship to add an elementary Step to a BreakdownList of tasks.

b. Functional Properties

If a property is *functional*, then, for a given element, there can be at most one element that is related to the element via the property; for example, $Role_n$ is responsible for $Artifact_m$. Another example that we have used for validation is the requirement to have Performer for Task through the relation TaskHasPerformers.

c. Inverse Functional Properties

If a property is *inverse functional*, it means that the inverse property is functional. So, if $Artifact_m$ is produced by $Role_a$ and $Role_b$ is responsible for $Artifact_m$, then $Role_a$ and $Role_b$ are the same; e.g. if the relation TaskHasPerformers is inverse, then that relationship is the same, whether it is the role or the task that is the active member, or, vice versa, the target of the relationship. An example of a non inverse relationship would be that between Task and WorkProduct, where the direction of the relationship indicates that WorkProduct is produced or consumed by Task through the relation TaskHasWorkProducts.

d. Transitive Properties

If a property **P** is *transitive*, and that property relates element **A** to element **B**, and also element **B** to element **C**, then element **A** is related to element **C** via property **P**. This relation is very important in the ability to draw up a BreakdownList according to a particular flow (Activity Workflow, Knowledge, Risk and Management). Through transitivity, we can deduce the order of the activities.

e. Symmetric Properties

If a property **P** is *symmetric*, and that property relates element **A** to element **B,** then element **B** is also related to element **A** via property **P**. The relation TaskHasPerform is symmetric, but the

relation TaskHasSteps is not (see Figure 3.1 for additional details). The first relation involves the deduction of roles by fixing Task, while the second involves Step cannot contain a Task.

f. Antisymmetric properties

If a property **P** is *antisymmetric*, and that property relates element **A** to element **B,** then element **B** cannot be related to element **A** via property **P**. For example, $Step_n$ is a partOfTask $Task_m$, and the relation partOfTask is *antisymmetric*, which means that $Task_m$ cannot be part of $Step_n$.

g. Reflexive properties

A property **P** is *reflexive* when that property must relate element **A** to itself. For example, WorkProduct can refer to itself with the relation isDeliverable.

h. Irreflexive properties

If a property **P** is irr*eflexive*, it can be described as a property which cannot relate an element **A** to itself. For example, $Task_a$ isPrecededBy itself. We have implemented this specification in the relation TaskReferencesTargetTasks, which is not reflexive, but rather irreflexive. The management of task sequencing is specified in SPEM by an enumerator (finishToStart, finishToFinish, startToStart, and startToFinish).

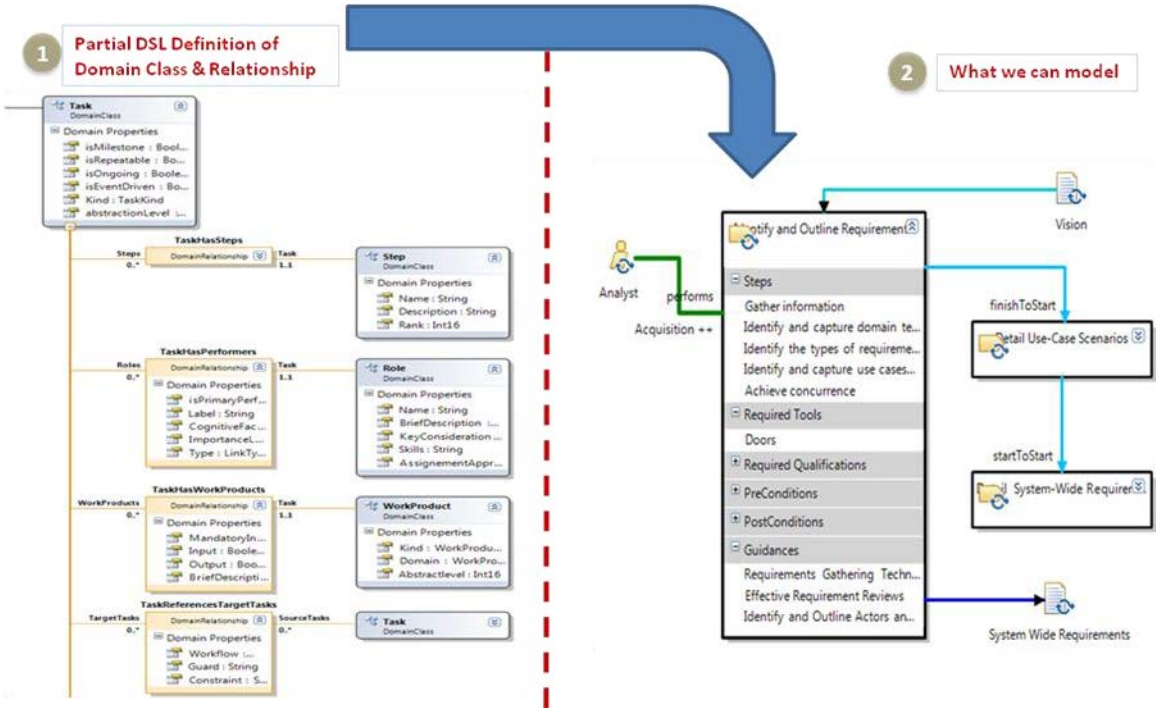


Fig. 3. 1- Partial conceptual model; 2- Modeling example

Figure 3 shows a partial view of the conceptual model (left-hand side). This view presents a Task class domain that is attributed, and its attributed relations with the classes Step, Role, WorkProduct, and Task. An example of the model that can be obtained is presented on the right-hand side of the figure. The domain classes, identified in blue, represent the SPEM elements. Each element either has attributes defined in the SPEM specification, or new ones, which were added to satisfy a given view (see Figure 7). For example, a Step has the attributes Name and Description, and a Rank, which means an order of operation.

Furthermore, the elements structuring a model process are linked by relations, identified in pink. Unlike UML, these relations also have attributes, in order to better qualify the nature of the relation. For example, the attributes of the relation TaskHasPerformers are elementary in type (e.g. label of type string) or of a type that refers to a class (e.g. CognitiveFac is of the CognitiveFactor type). It should be noted that the relations that have no attributes (e.g. TaskHasSteps, left-hand side of Figure 3) are specific relations of membership, and in this example the object Step cannot exist apart from a Task. For example, the relation TaskHasPerformers identifies the characteristics of the link between a Task and a Role, it tells us whether or not the role is primary, as well as what cognitive factor is required for the realization of the task and its importance. The relation TaskReferencesTargetTasks defines the nature of the sequencing between the tasks as specified in SPEM (finishToStart, finishToFinish, startToStart, startToFinish). The right-hand side of the figure shows an example of what a process engineer can design. The details of the tool are illustrated in the following section.

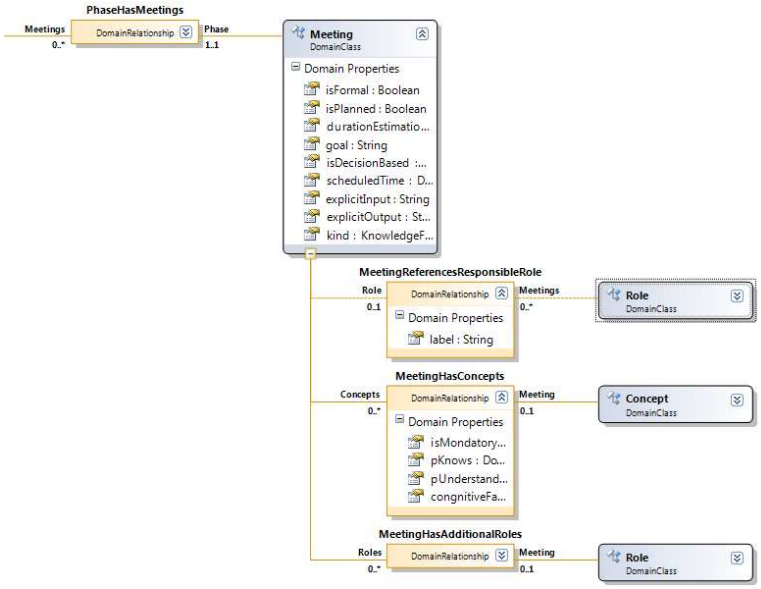


Fig. 4. 1- Example of "Meeting" element integration that is not provided by the SPEM specification

DSL enables us to add new classes and new relations to allow modeling that is adapted to the specific needs of a process model. For example, we illustrate the case where the aim is to model meetings within the process model. The domain class “Meeting” has attributes which make it possible to physically characterize the meeting and its relationships to role and concepts. The domain class “Concept” allows representation of the required or desirable knowledge elements that will be addressed at the meeting.

Figure 4 illustrates the integration of the “Meeting” class into the conceptual model, which is not specified in the SPEM. This element is generally linked to a form of important tacit knowledge within development project activities (planning, establishing and maintaining a shared vision of the problem being solved, post-mortem analysis, etc.). The relation PhaseHasMeetings indicates that one can have one or more “Meeting” objects in a phase. These “Meetings” are characterized by attributes (isFormal, isPlanned, durationEstimation, goal, isDecisionBased, etc.) and must be related to “Roles” and “Concepts”. A concept defines the type of knowledge involved in the meeting, examples of which are: knowledge of the business, knowledge of the tests, knowledge of technical practices, management expertise, etc. The relation MeetingHasConcepts relays the importance of the use of the concept within the framework of the meeting (isMandatory) and on the nature of the knowledge (Knows or Understands). These two attributes indicate the level of knowledge or of comprehension of the concept. The attribute CognitiveFactor indicates the nature of the cognitive activity required to grasp this concept at the meeting: acquisition, synchronization, crystallization, validation.

4. Description and characteristics of the tool

A PCSE called DSL4SPEM (Domain-Specific-Language for Software Process Modeling) has been created to support our PML approach. The tool is a Microsoft Visual Studio 2008 Plug-in, and satisfies the meta-requirements for PMLs and PCSEs defined and revised by many authors [3, 8, 14, 28-32].

This tool gives process engineers the ability to graphically design process models (see Figure 5) in the same way that software designers design applications. Using the same SPEM elements instantiated in the graphics scene, the designer can instantiate one or more relations between those elements. Every relation can contain several classes of attributes to represent a given problem.

Tool illustration

In order to validate the benefits of this approach and the performance of the DSL4SPM tool, a maintenance process has been modeled in conformity with the ISO/IEC 14764 2006 standard [7]. The partial result is shown in Figure 5.

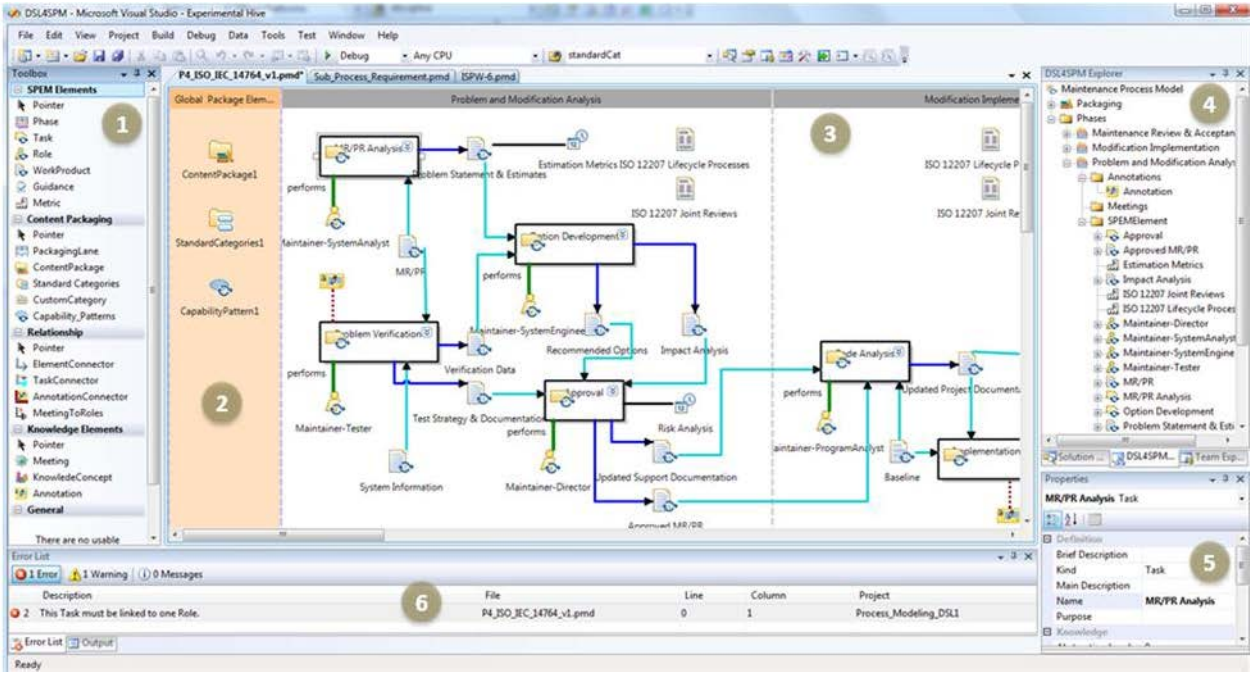


Fig. 5. DSL4SPM modeling interface

Figure 5 presents the DSL4SPM modeling environment. This environment is organized into six main zones:

- Zone 1 is the "Toolbox", which contains the instantiable elements classified in groups, as follows:
  - SPEM elements (those specified in SPEM). For example, the drag-and-drop element of the task icon in Zone 3 allows instantiation of a task (e.g. "Problem Verification"). The sub-elements of a task can be specified by right clicking on the box defining the task. Examples of sub-elements are presented on the right-hand side of Figure 3 (e.g. components of a task: steps, required tools, pre-condition, post-condition, guidelines).
  - Content packaging: includes elements, specified in SPEM, which have a more global range that can satisfy both a need for generalization and a need for reuse. They are instantiable only in the Swim Lane "Global Package Elements" (Zone 2), which is unique to the process model. Below, we provide a short description of these elements. For additional details, please consult the SPEM specification available on the OMG website:

- ✓ ContentPackage: a container for reusable elements, such as role definition and guidance. Note that a direct connection to "ContentPackage" from EPF Composer is envisaged to recover the reusable concepts.
- ✓ Standard Categories: a container for categories for classifying SPEM elements. For example, Task can be classified in Discipline and Domain, and Role in RoleSet. DSL4SPM makes it possible to import, in XML format, classifiers such as those used in the OpenUP and Scrum processes.
- ✓ CustomCategory: for creating other types of classification not available in Standard Categories. For example, the designer can add a category which respects the VSTS definition of WorkItems.
- ✓ Capability Patterns: a container for parts of processes to be put into a dynamic process assembly which is based on reuse. This element will be involved in future work, which will await definition of the operators required to realize this type of dynamic assembly.

- Relationship: the types of relations that the designer can use to link two elements of the scene. A major effort has been made to simplify the number of relations in ToolBox. The system automatically detects the original and target elements, and instantiates the type of relation required.
- Knowledge elements: elements added specifically to satisfy a view based on the management of knowledge (both tacit and explicit). Note that the element Annotation, borrowed from the cognitive maps domain, has also been added to facilitate collaboration during process model design. This collaboration takes the form of a text message and can be one of the following: Advice, Change, Comment, Example, Explanation, Question, or See Also.

The addition of new objects, borrowed from other disciplines, shows the type of enrichment that can be brought to the process modeling domain by the use of models.

- Zone 2 comprises the contents of the global elements. These elements originate from Content Packaging in Zone 1.
- Zone 3 is the modeling scene, structured in Swim Lanes, which represents the phases of the process. Each element or relation is constructed using drag-and-drop from Zone 1. The icons in Zone 3 can take various forms, depending on the possible categories for the element in question (e.g. Work Product can have one of the following icons: Artifact, Deliverable, Outcome, TechnicalArtifact).
- Zone 4 is DSL4SPM Explorer, which constitutes the data repository of the model being created. This repository is stored in a single XML file, which facilitates collaboration, as well as

modification and management of the versions. Below, we illustrate the potential of this repository to display the various modeling perspectives.

- Zone 5 displays the properties of each instantiated object in the scene. These properties are organized according to the corresponding predefined views. The modeling elements presented in Zone 1 are generic and can be specified by distinctive icons.
- Zone 6 is dedicated to posting exceptions and error messages sent by the system following a request for validation of the model. The validation is performed in accordance with predefined rules of coherence, such as the necessity for a principal role for realizing a task.

The core principle of the DLS4SPM is the use of a toolbox containing basically SPEM elements. Once the objects have been instantiated, the designer is able to link them with a type of relation and define the values of the attributes necessary for the objects and the relations, with the aim of consolidating a particular view.

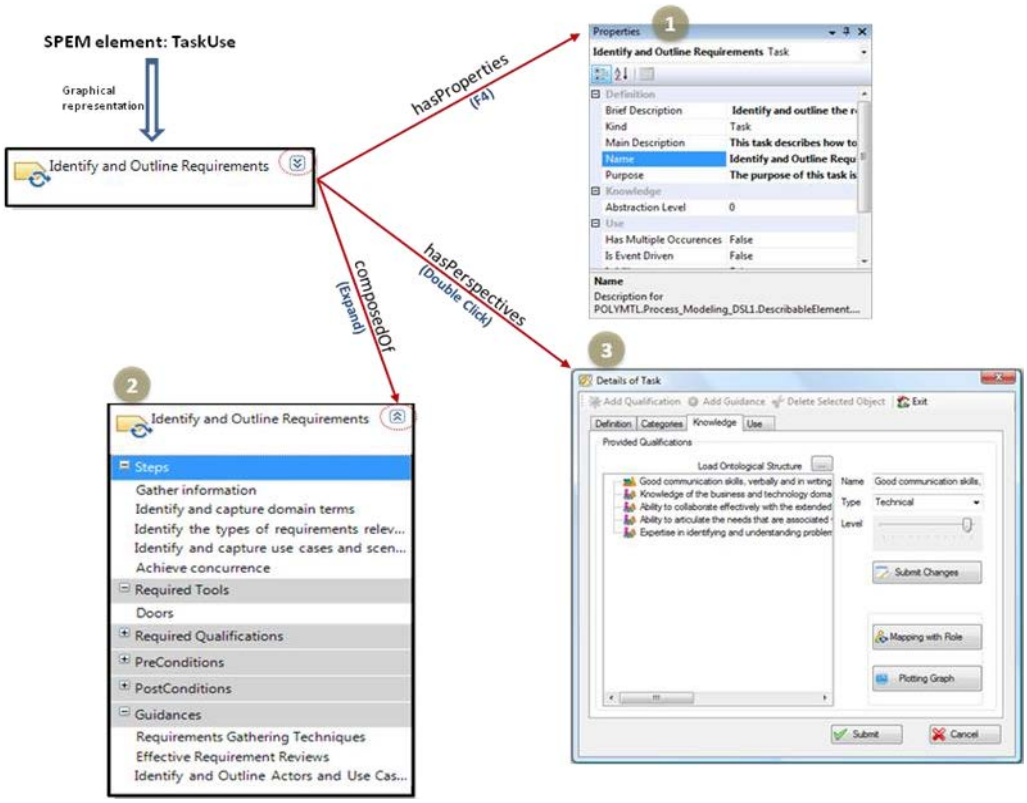


Fig. 6. The various levels of abstraction for representing a SPEM element

Figure 6 shows the conceptual approach to the representation of SPEM elements in a process model. This approach adopts three levels of abstraction, according to the degree of accuracy targeted by the process engineer.

- 1- First level of detail, illustrated by the arrow “hasProperties”: permits the definition of a set of properties which characterizes an object in the scene. These properties are grouped together in order to facilitate the representation of a given view. For example, a Task can be instantiated by indicating the name in the properties window (e.g. Identify and Outline Requirements).
- 2- Second level of detail, illustrated by “composedOf”: makes it possible to add the sub-elements specific to the Task (Steps, Required Tools, Required Qualifications, Pre-condition, Post-condition). Each sub-element is also characterized by a list of properties (e.g. a “STEP is also characterized” has the properties: name, description and an order of realization). This approach significantly simplifies the number of elements in ToolBox.
- 3- Third level of detail, illustrated by the arrow “hasPerspectives”: indicates the properties of the task in order to register it in a particular flow, such as “Knowledge” for the cognitive aspect and “Uses” for the use aspect of the task. In this case, we preferred to use a personalized form to provide more flexibility on the level of mapping between the element in question and the other elements of the scene which are connected to it.

The cognitive aspect (Third level in figure 6) refers to an XML file containing the list of required competencies for realizing a task, and a mapping is made automatically with the list of skills provided by the role. The aim is measurement of the variation in knowledge between the role and the task. The usability aspect ensures mapping to the VSTS or MS Project. Discussion of these two aspects is outside the scope of this paper.

Figure 7 shows the view generation functionality from the data repository which is stored in standard XML format. DSL4SPM Explorer (Zone 4 in Figure 5), illustrated on the left-hand side of the figure, lists all the elements of the modeling scene. These elements are organized in phases. For example, the “Problem and Analysis Modification” phase (third from top of the phases list) contains an annotation, two planned meetings, and lists of SPEM elements. The Breakdown Structure, based on the activities view according to the flow of activity, is addressed (right-hand side of the figure). It is possible to address other views (Knowledge, Risk, Organizational, Human) based on specific attributes. This concept of views allows the process designer to focus on the interests concerned.

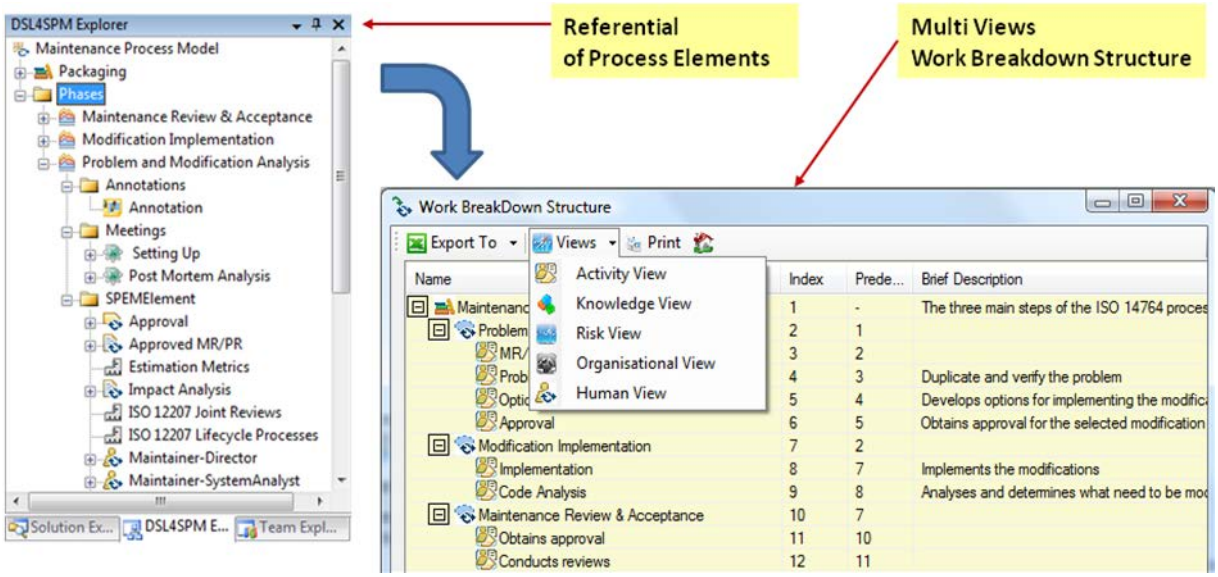


Fig. 7. Generation of a number of views based on the same process model

5. Concluding remarks

This process modeling approach and the DSL4SPM tool have already been used by several different groups in the context of software process training. We have observed that this approach is intuitive and leads to fast and effective software process modeling. The integration of the DSL4SPM tool into an Integrated Development Environment (IDE) offers software process engineers a natural use for the tool.

The concept of DSL4SPM and its proof of concept implementation into a tool was realized to bring a closer integration between the software process model and the two dimensions of the software development environment, which are the appropriateness of the process model regarding the practices and the integration of the process model into the development environment used by the developers. The following five aspects are what we believe to be the most important contributions toward this objective:

- The DSL component enables the modeling of hybrid processes, in which the disciplined and agile practices are combined. Collaborative activities such as pair-programming, stand-up meeting and design meeting can be modeled at the various phases of the process. They are defined in terms of outcomes or objectives rather than in terms of artefacts as with conventional process modeling.

- The ontology definition validates the coherence rules of a process model and provides helpful guidance when the model element relationships do not conform to the ontology rules. It enables diagnosis at the conceptual level of modeling. For example, the maintenance process based on the ISO/IEC 14764 2006 standard, which is presented in this paper, identifies only a single role called 'Maintainer', while the modeling of this process according to DSL4SPM shows that various competences are required to fulfill all the tasks and accordingly more than a single role is needed.
- The feature, which enables the definition of customized attributes for each SPEM element, makes it possible to visualize the model according to various perspectives. For example, the traditional activity perspective provided by most software process models can be complemented with a risk level perspective or a knowledge flow perspective.
- Some implementation related features facilitate the integration of the DSL4SPM tool to IDE. One basic implementation feature is to record modeled processes on XML files, which facilitates model configuration management and model exporting.
- A pragmatic software process model should be exportable to various software tools for detailed process element descriptions or integration into a development environment. For example, processes modeled in DSL4SPM can be exported to content manager like EPF Composer for a textual description of all the elements making up the process. The process model can also be exported to VSTS to contribute to the definition of tasks, thereby facilitating the management of the draft agreement with the real process in-used. Exporting the process model tasks to MS Project tool facilitates resource planning in accordance with the software process model.

There are conceptual and technical limitations to this approach. Conceptually, there is no limit to the number of different views or perspectives that can be modeled. A new perspective is based on the attributes that are associated to the relationships within the model. Lack of rigor in informally defining these attributes can lead to misleading perspectives. In complex software process modeling, which means including a large number of elements, it becomes difficult to manage many perspectives.

The DSL4SPM proof of concept tool was implemented as a Plug-in in the IDE 'Visual Studio'. Of course, this limits the use of the DSL4SPM tool to that environment. This environment has been used for two major reasons: the DSL Tool Framework is provided has ready to use graphic support

and the C #partial class mechanisms facilitate implementation of such a tool. However, this is a weak limitation, since given appropriate resources this tool could be implemented in any environment.

This project presents a new approach to software process modeling, based on SPEM for the syntactic definition and on DSL for the semantic definition of a process model. These two definition components represent structural and behavioural views respectively. A DSL4SPM tool has been created as a proof of concept to demonstrate the benefits of this approach.

With DSL4SPM, the process engineer is able to quickly and easily model a process from an innovative perspective, with the aim of identifying the prospective views associated with the software development project. These views can be created by the process engineers by defining the attributes of the elements and relationships required to define them. Examples would be: the activity flow view, the knowledge flow view, the risk view, etc. This can be done according to formal ontological concepts.

Future work will include a more detailed analysis of the concepts related to modeling the knowledge flow between the Role and Task components, with the aim of evaluating the propagation of tacit knowledge through all phases of the process. Tacit knowledge complements the explicit knowledge embodied in the artifacts, and this will enable us to integrate an important concept from an agile process into a disciplined process in a rational way.

Acknowledgments

We would like to express our thanks to Mathieu Lavallee who was deeply involved at the technical level to carry out the DSL4SPM tool.

This work was partly supported by the “Fonds de Recherche sur la Nature et les Technologies” council of Québec (FQRNT) under Grant 127037.

References

[1] Eclipse.org, "EPFComposer," http://www.eclipse.org/epf/, visited 10 February, 2009.

[2] P. N. Robillard, N. Kerzazi, M. Tapp, and H. Hmima, "Outsourcing software maintenance: processes, standards & critical practices," in *2007 Canadian Conference on Electrical and Computer Engineering*, Vancouver, BC, Canada, 2007, pp. 682-685.

[3] P. H. Feiler and W. S. Humphrey, "Software process development and enactment: concepts and definitions," Los Alamitos, CA, USA, 1993, pp. 28-40.

[4] G. Booch, A. Brown, I. Iyengar, J. Rumbaugh, and B. Selic, "An MDA Manifesto," in *http://www.bptrends.com/publicationfiles/05-04%20COL%20IBM%20Manifesto%20-%20Frankel%20-3.pdf*, M. Journal, Ed., 2004.

[5] OMG, "Software Process Engineering Metamodel Specification, SPEM v2.0," *http://www.omg.org/spec/SPEM/2.0/PDF*, 2008.

[6] S. Cook, G. Jones, and S. Kent, *Domain-specific Development With Visual Studio Dsl Tools*, Addison-Wesley ed., 2007.

[7] ISO/IEC, "Software Engineering — Software Life Cycle Processes — Maintenance," *International Standard ISO/IEC 14764* 2006.

[8] S. M. Sutton and L. J. Osterweil, "The Design of a Next-Generation Process Language," in *ESEC/ACM Foundations of Software Engineering Symposium*, 1997, pp. 142-158.

[9] M. I. Kellner, "Representation formalisms for software process modeling," in *`Representing and Enacting the Software Process'. Proceedings of the 4th International Software Process Workshop (Cat. No.88TH0211-3)*, Moretonhampstead, UK, 1989, pp. 93-96.

[10] W. Deiters and V. Gruhn, "Managing software processes in the environment MELMAC," USA, 1990, pp. 193-205.

[11] S. Bandinelli and A. Fuggetta, "Computational reflection in software process modeling: The SLANG approach," Los Alamitos, CA, USA, 1993, pp. 144-54.

[12] N. Belkhatir and W. L. Melo, "Supporting software development processes in Adele 2," *Computer Journal,* vol. 37, pp. 621-638, 1994.

[13] R. Conradi, B. P. Munch, J. O. Larsen, M. N. Nguyen, and P. H. Westby, "Integrated product and process management in EPOS," *Integrated Computer-Aided Engineering,* vol. 3, pp. 5-19, 1996.

[14] X. Junchao, L. J. Osterweil, Z. Lei, A. Wise, and W. Qing, "Applying little-JIL to describe process-agent knowledge and support project planning in soft PM," *Software Process: Improvement and Practice,* vol. 12, pp. 437-448, 2007.

[15] B. Curtis, "Three problems overcome with behavioral models of the software development process," Piscataway, NJ, USA, 1989, pp. 398-399.

[16] E. Di Nitto, L. Lavazza, M. Schiavoni, E. Tracanella, and M. Trombetta, "Deriving executable process descriptions from UML," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, Orlando, FL, 2002, pp. 155-165.

[17] K. Z. Zamli and P. Lee, "Exploiting a virtual environment in a visual PML," in *Product Focused Software Process Improvement. 4th International Conference, PROFES 2002. Proceedings*, Rovaniemi, Finland, 2002, pp. 49-62.

[18] P. Kruchten, *The Rational Unified Process: An Introduction (3rd Edition)*: Addison-Wesley, 2003.

[19] C. Gonzalez-Perez and B. Henderson-Sellers, "A comparison of four process metamodels and the creation of a new generic standard," *Information and Software Technology,* vol. 47, pp. 49-65, 2005.

[20] ISO/IEC, "Software Engineering — Metamodel for Development Methodologies," *International Standard ISO/IEC 24744 First edition,* 2007-02-15.

[21] C. Gonzalez-Perez and B. Henderson-Sellers, "Modelling software development methodologies: A conceptual foundation," *Journal of Systems and Software,* vol. 80, pp. 1778-1796, 2007.

[22] A. Gavras, M. Belaunde, L. F. Pires, and J. P. A. Almeida, "Towards an MDA-based development methodology," in *Software Architecture. First European Workshops, EWSA 2004. Proceedings*, St Andrews, UK, 2004, pp. 230-240.

[23] F. Jouault and J. Bezivin, "KM3: a DSL for metamodel specification," in *Formal Methods for Open Object-Based Distribution Systems. 8th IFIP WG 6.1 International Conference, FMOODS 2006. Proceedings*, Bologna, Italy, 2006, pp. 171-185.

[24] J. Greenfield, K. Short, S. Cook, and S. Kent, *Software factories: assembling applications with patterns, models, frameworks and tools*: Wiley, 2004.

[25] J. Bezivin and E. Breton, "Applying the basic principles of model engineering to the field of process engineering," *UPGRADE: The European Journal for the Informatics Professional,* vol. 5, 2004.

[26] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *International Journal of Human-Computer Studies,* vol. 43, pp. 907-928, 1995.

[27] M. Horridge, et al., "Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools: 2007," http://www.co-ode.org/resources/tutorials/protege-owl-tutorial.php, visited 10 November 08.

[28] S. Bandinelli, A. Fuggetta, and S. Grigolli, "Process modeling in-the-large with SLANG," Los Alamitos, CA, USA, 1993, pp. 75-83.

[29] R. Conradi and L. Chunnian, "Revised PMLs and PSEEs for industrial SPI," in *Object-Oriented Technology. ECOOP'97 Workshop Reader. ECOOP'97 Workshops. Proceedings*, Jyvaskyla, Finland, 1998, pp. 289-294.

[30] K. Z. Zamli and P. A. Lee, "Taxonomy of process modeling languages," in *Proceedings ACS/IEEE International Conference on Computer Systems and Applications*, Lebanon, 2001, pp. 435-447.

[31] S. Arbaoui, J. C. Derniame, F. Oquendo, and H. Verjus, "A comparative review of process-centered software engineering environments," *Annals of Software Engineering,* vol. 14, pp. 311-340, 2002.

[32] R. Bendraou, M. P. Gervais, and X. Blanc, "UML4SPM: A UML 2.0-based metamodel for software process modeling," in *Model Driven Engineering Languages and Systems. 8th International Conference, MODELS 2005. Proceedings*, Montego Bay, Jamaica, 2005, pp. 17-38.

L'École Polytechnique se spécialise dans la formation d'ingénieurs et la recherche en ingénierie depuis 1873

École Polytechnique de Montréal

École affiliée à l'Université de Montréal

Campus de l'Université de Montréal
C.P. 6079, succ. Centre-ville
Montréal (Québec)
Canada H3C 3A7

www.polymtl.ca

ÉCOLE
POLYTECHNIQUE
MONTRÉAL