Using component-oriented process models for multi-metamodel applications

Fahad Rafique Golra, Fabien Dagnat

Using Component-oriented Process Models for Multi-Metamodel Applications

Fahad R. Golra
Université Européenne de Bretagne
Institut Télécom / Télécom Bretagne
Brest, France
Email: fahad.golra@telecom-bretagne.eu

Fabien Dagnat
Université Européenne de Bretagne
Institut Télécom / Télécom Bretagne
Brest, France
Email: fabien.dagnat@telecom-bretagne.eu

***Abstract*—Recent advancements in Model Driven Engineering (MDE) call for the corresponding software development processes. These processes need to be able to assist multi-metamodel development by extending support for the usage of models and transformations. Business processes are sequenced on the basis of their sequential contingencies whereas we argue that software development processes are meant to be sequenced along their intrinsic factors. In this paper, we present a process metamodel from our framework, inspired from the component based paradigm, to automate the software development processes. This approach presents the concept of structured artifacts and exploits the activity sequencing based on events and constraints.**

I. Introduction

The recent progress of Model Driven Engineering (MDE) has shaped the software industry to be model centric. Development through the evolution of one model from requirements till deployment passing through a series of transformations is the aim of MDE [1]. The progress towards the achievement of this goal needs corresponding process modeling support, which has been quite overlooked [2].

The factors restraining software industry to unleash the full potential of Model Driven Engineering are quite explored [3]. One of these important factors is the lack of coherence between software process modeling and software development paradigm. To achieve coherence amongst the model driven development paradigm, software development lifecycles and software development processes, we argue that the software development process modeling should also be model centric. Besides this, a process modeling approach should allow the flexibility for process improvement, not only at organizational level but also within a project. This can be achieved if the processes can be replaced or updated without affecting their context.

Process modeling domain is dominated by business process models that focus on the workflow and sequence of the processes [4]. This results in the lack of appropriate mechanisms for validation, verification and state maintenance of the work products. Software process modeling being targeted for the software industry should match the perspectives of the domain experts. The software jargon is more familiar with execution, initiation, implementation, and typing. We argue that current software process models are more close to business process modeling which creates a gap between the process modeling and architecture modeling. In order to reduce this gap, software process modeling approach is tailored to a software development paradigm (Component based software engineering), that is more comprehensible to software domain experts. The choice of component based paradigm helps in developing process components with specified interfaces. This approach favors both the execution mechanisms and the process improvement.

This paper is structured as follows. Section 2 describes the recent endeavors in the field of software process modeling. Section 3 describes the General Process Metamodel. Section 4 presents the complete approach as in a Multi-metamodel framework. Finally Section 5 outlines the conclusions.

II. Software Process Modeling

Various approach have been proposed to model and *execute* processes. This section describes the most notable approaches.

SPEM2.0 is presented by OMG as a standard with the vision of separation between the usage and the content for the software development processes [5]. Different UML diagrams are used to model different views of the process model. The usage of these diagrams adds expressibility to process model but as a side effect, the model under study faces semantic limitations. These semantics limitations can vary from inter-process communications, artifact definitions, event descriptions to execution. Lack of proper explanation for exception handling leads to the absence of reactive control for the processes. In terms of execution, process enactment is nothing more than a mapping to project plan. This offers no support even for process simulations. The process components introduced by SPEM2.0 lack their proper semantics as well. These process components take WorkProducts as ports of a component, thus linking process components together on the basis of these WorkProducts. Keeping SPEM as a standard, various other techniques tend to extend it in order to overcome its shortcomings.

OPSS is a workflow management system that is implemented on top of Java Event-based Distributed Infrastructure (JEDI) to use its event-based approach [6]. OPSS presents a translator that automatically translates the UML process

models to java code. OPSS then enacts the process by executing this java code. The architecture of OPSS revolves around two main components as agents and the state server. The State server is responsible for managing the state of the processes, which helps in the coordination of agents. An event notification system defines a standard inter-operation mechanism between agents. Besides the proactive control, the use of events in state transitions offers a reactive control in activity sequencing. As events are used to trigger transitions thus it is also possible to define error states leading to exception handling. This approach adds up to the semantics of UML at a lower level by translating it to java code.

Chou's method is a process modeling language that consists of two layers: high level UML-based diagrams and a corresponding low level process language [7]. The high level UML based diagrams use a variation of activity and class diagrams, whereas the low level process language is object oriented language and models a process as a set of classes (the *process program*). The high level diagrams hide the complexity of the rich semantics at the lower level. These process classes exploit exception handling and event signaling to offer reactive control. An automatic mapping between the two levels is not provided (with some tool), however their correspondence is well documented. This approach shares a common drawback with OPSS, that is to be complex. The software industry did not welcome the complex modeling approaches (model programs) presented in past decade.

MODAL is a more recent work, enriching the original metamodel of SPEM to exploit the potential of Model Driven Engineering [8]. A concept of *intention* is introduced in MODAL to keep track of methodological objectives set by different stake-holders from an activity. Contrary to SPEM, it focuses more on execution of the process models, thus the ports of the process components are not taken up as work products, rather they act as services i.e. much like the service ports in component based programming paradigm. These process components are setup as hierarchical abstraction levels: Abstract Modeling Level, Execution Modeling Level and Detailed Modeling Level, which describe the process components from coarse grained analysis to fine grained analysis. SPEM does not provide the reactive control over the process sequence, thus a flexible constraint application offered by MODAL can help in developing more stable process models.

UML4SPM is a recent endeavor for achieving executability for process models [9]. It is presented using two packages: the Process Structure package which defines the primary process elements and the Foundation package that extends a subset of the concepts from UML, specifically tailored for process modeling. Sequence control is offered at two levels (Activity sequencing and Action sequencing) through control flows and object flows. Actions serve as basic building blocks for the activities. This control is defined in terms of control nodes. A strong proactive control in the language is ensured by the use of these control nodes along with action and activity sequencing. An activity has the ability to invoke an action that can alter the control sequence of activities, thus offering

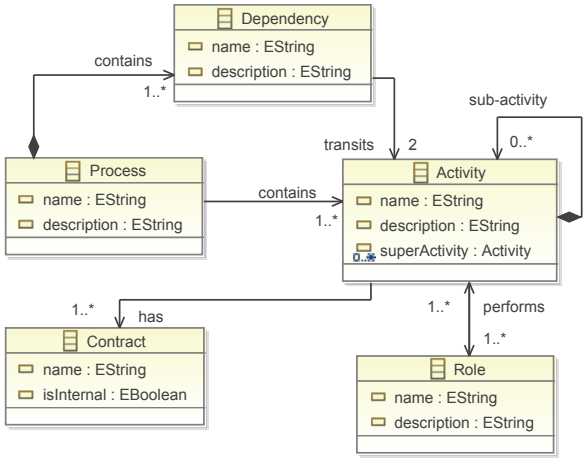

Fig. 1. Core Package

reactive control. Though they take full advantage of MDE for the specification of their own approach, their approach is not targeted towards the support of model driven software development.

xSPEM stands out amongst all the process modeling languages discussed earlier in terms of execution semantics [10]. This approach extends SPEM with project management capabilities, a better process observation and event descriptions. This gives a concrete basis for the process execution. A solid tool support is provided by a graphical editor and a model simulator. xSPEM also offers model validation by translating the properties on SPEM into LTL properties on the corresponding Petri Net model. Project monitoring is handled by a mapping to BPEL.

All these approaches tend to support enactment but they do not exploit the full potential of MDE through the use of models. An approach where model is the basis of communication between the processes is still missing. Not much work has been done on the process modeling approach where models can serve as the input and output artifacts of a process. Though many current approaches offer processes as components, they fail to provide the execution semantics for these process components. Most of the process modeling approaches use SPEM as their basis and tend to add up to its expressibility by translating the model into a process program or complementing the model with a process program. A concrete approach for a semantically rich process modeling language is still missing.

III. General Process Model

General Process metamodel defines all the basic structure of our framework. It is presented using three packages where activity implementation package and contract package merge into the core package. The core package defines the five core entities of our process framework, as illustrated in Figure 1. A process is defined as a collection of activities that can be arranged in a sequence through dependencies based on their contracts. Instead of focusing on the proactive control for the

process, more focus is given to activity being a basic building block for the process. A process having both activities and their associated dependencies represents an architecture using activities as basic entities and dependencies to define the flow between them.

An activity is decomposable thus presenting the process model as a hierarchical structure. Activities can be shared amongst different processes and super activities. An activity behaves as a black box component, where the interface to its context and content (in case of composite activity) is through its contract. Inspired from the component based paradigm, all the interactions between two activities is handled through this contract. Each activity is performed by role(s). No explicit control flow is defined for the activities. The contracts of the activities and the dependencies between them together allow the dynamic sequencing of flow for the processes. This dynamic sequence of processes gives a reactive control for process models that have the capability of restructuring the control flow at runtime.

Another package called the activity implementation package defines the content of the activity, as shown in Figure 2. As recently adopted by SPEM2.0, the current approach is to separate usage from the content. We have also used this approach for the definition of activity. Each activity is implemented by one or more activity implementations. These activity implementations present a set of different alternative usages. One amongst these implementations can be used later on to decide the process instance. A contract that serves as an interface to the context or content is associated to each activity implementation. An activity implementation has to conform to the contract of its activity (type). This contract is used by the dependencies that serve for the transition of control flow at the time of execution.

An activity implementation can either be primitive or composite. In case of a composite activity implementation, it contains a process. A composite activity containing a process, encapsulates the activities and dependencies contained by that process, which serves as its content. In order to interact with the content, this composite activity implementation uses the internal contracts. All the interactions of the contents of this composite activity to its context has to pass through this activity. In case of a primitive activity, its implementation defines the procedure for its realization.

A primitive activity can be automatic, semi-automatic or manual. All activities are performed by roles. If no performer is amongst the associated roles, then the activity is automatic. If no tool is amongst the associated roles, then it is a manual activity. In case of a semi-automatic activity a performer performs the activity by using a tool. Such a categorization of activities helps manage different types of activities where special focus is given to automation of activities in order to use model transformations.

The third package of the general process metamodel is the contract package as shown in Figure 3. Activities can be connected together for the purpose of control flow only using these contracts. A Contract may be internal or external. In case of composite activities, for each external contract there is a corresponding internal contract of the opposite nature. A nature of a contract is either required or provided. Thus for a required external contract, there exists a provided internal contract and vice versa. Each contract defines the events, properties and artifacts, so that the dependency can link together two activities. These events, properties and artifacts can be optional or mandatory, based on the behavior of the activity.

An event in our framework can be a triggering event, a consumable event or a producible event. A triggering event is responsible to trigger an activity. Every required contract defines the triggering events for the activity, which are responsible to trigger its initiation. Provided contracts do not have any triggering event. A consumable event is an event that may be consumed by an activity for its realization. If the consumable event is mandatory, the activity can not be performed without this event, however an optional event can be neglected. Consumable events are defined in the required contract of the activities. A producible event is defined in the provided contract, which may or may not be mandatory. This event is produced as a target or side effect of the realization of the current activity. This provided event may be used by the successor activity as a required event.

An artifact defines the resource needed or produced by an activity. An artifact in our process framework can either be a model or any other resource. In case of a model, an artifact defines the metamodel for the contract. This adds up the flexibility to use models as a basis of communication between the activities. Such models can be used for the semi-automatic transformations in order to carry out model transformations. A needed resource is a requisite which is defined in the required contract whereas a produced resource is defined in the provided contract as a deliverable. An artifact can be optional or mandatory, depending upon the nature of the activity.

The contracts of an activity also define the properties that help linking two activities together through the use of dependencies. There are three types of properties i.e. invariants, pre-conditions and post-conditions. Pre-conditions are defined in the required contract of an activity. These are used to evaluate the properties, that need to be met in order to start the realization of the activity. Invariants are the properties that need to be met throughout the execution of an activity and are thus also defined in the required contract. Post conditions on the other hand are defined in the provided contract of an activity and record the conditions created by the realization of an activity. The contracts of the sub-activities conform to the contracts of the super-activities.

One of the major benefits of using the component based approaches for process modeling, is to restrict the interaction through the specified contracts. Having defined specified contracts for the activities and activity implementations, we have the flexibility to choose any of the alternatives in activity implementations at runtime. Besides this, any activity implementation of an activity (type) can be replaced by any other alternative to work in the same context. This serves both

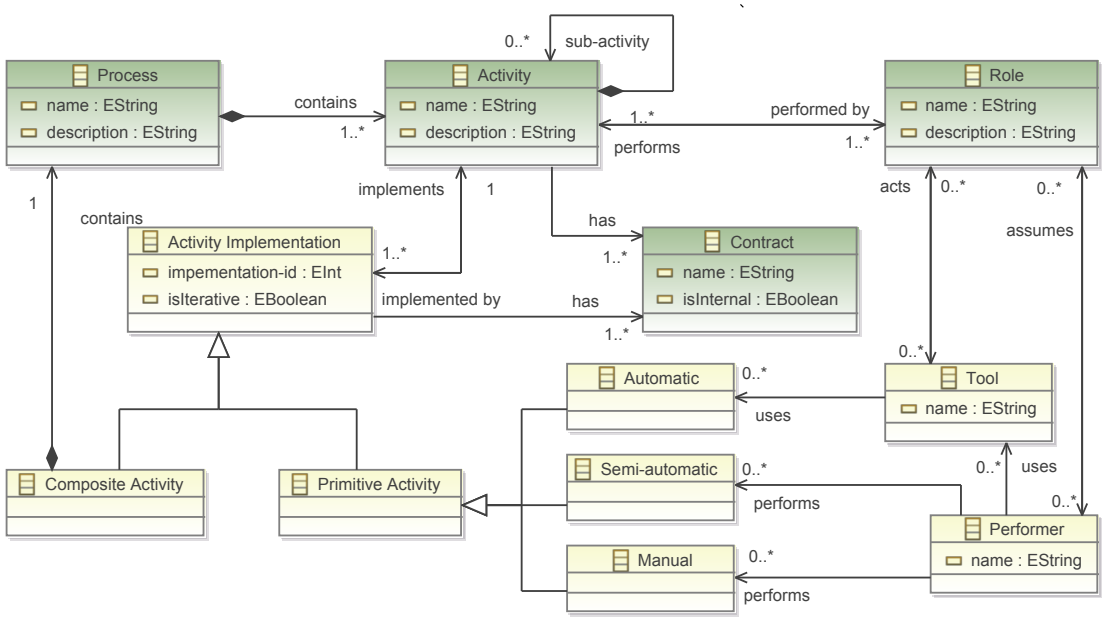

Fig. 2. Activity Implementation Package

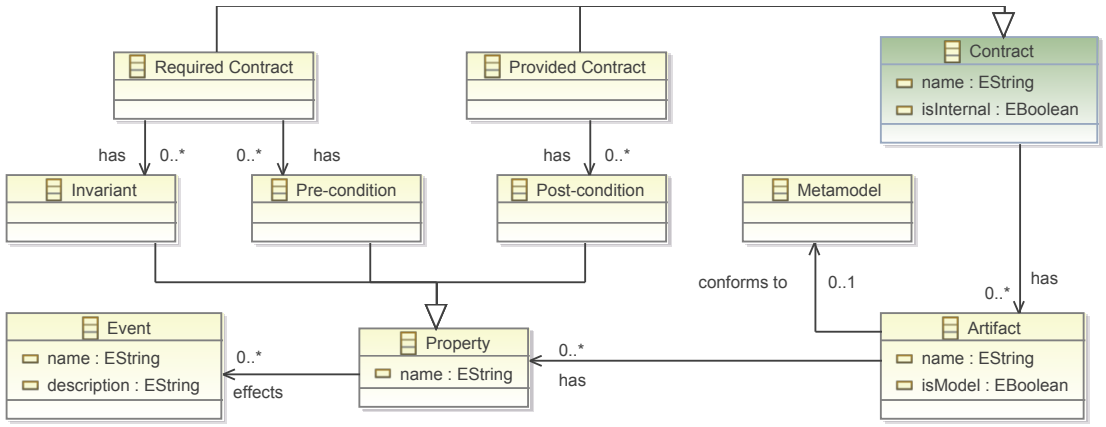

Fig. 3. Contract Package

for the content and context of an activity. Replacing one of the sub-activity implementations for an activity, does not affect its interaction with its context. Same way, some modification in the context, does not affect the interaction with the content of an activity.

IV. Multi-metamodel processes

The hallmark of MDE is the usage of multiple models along with defined transformations amongst them. Models are created, modified, merged or split, as the software development project advances. We argue that a unique process model cannot capture all the semantics of the processes at different development stages. For this reason, our approach presents three metamodels: the *General Process Metamodel*, the *Application Process Metamodel* and the *Instance Process Metamodel*. The *General Process Metamodel* is used to document the process best practices. It is not specific to certain organization or project. When this *General Process Metamodel* is applied to a specific project by some organization, it is refined to guide the development process. This project specific metamodel is called *Application Process Metamodel*. The *Instance Process Metamodel* is responsible for the execution of the processes for a project, thus it takes into account the time plan and status of the project. Model transformations are used between the models conforming to these metamodels. We are looking forward to provide the tool support for carrying out these transformations. This global picture is illustrated in Figure 4.

The development of these three process modeling metamodels along with the defined transformation definitions would allow us to create a complete framework. We have only presented the first metamodel i.e. *General Process Metamodel* in this paper, however the *Application Process Metamodel* and *Instance Process Metamodel* are underdevelopment. The Application Process Metamodel refines this metamodel with some additional features so as to make it application specific.

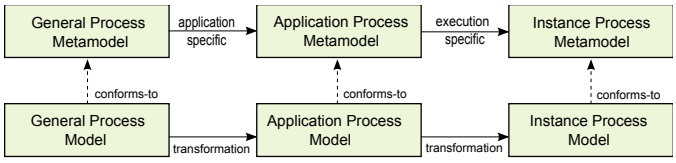

Fig. 4. Process metamodels for Multi-metamodel development

A categorization of activities is added to group them in disciplines. Activities can also be categorized on other aspects like tool collections and role collections. Guidance is used to guide each activity. In order to add planning, processes are then refined to phases and sub-phases, where each of them has a milestone.

Finally the Instance Process Metamodel is used to guide one specific process model instance. This metamodel refines the Application Process Metamodel so as to add up the capabilities to manage time. Actual project data (time-line, resource utilization, *etc.*) can be compared and analyzed against the expected data. This gives the ability to use the reactive control for process sequencing to reschedule the processes accordingly.

In our approach, we use model transformations between these metamodels. The vision of this software process modeling framework is that an organization can keep its general software development process models in the form of best practices acquired from the experience. These models can then be transformed to application specific models when some specific application is under development. Finally this application process model can be transformed to instance process model that gives the support for the execution of process models.

The tool for this software process modeling framework is visioned to be developed on top of METADONE, a METACASE tool providing a graphical environment for metamodeling [11]. The instantiation semantics for these processes is being sought, which would allow us to have an executable software development process modeling approach that can support model driven engineering. Furthermore we are looking forward for defining a formal semantics of these processes. Such formal semantics would allow us to verify the properties of the processes all way starting from the requirements phase till the deployment.

V. Process Example

In order to facilitate the understanding of our approach, we are modeling a process example that is responsible for developing a transformation definition and executing it to get the desired output model. This process is named as *Transformation Development* Process. For the purpose of clarity we are presenting here a graphical notation, which is only intended for the explanation purposes. It should be noted that our approach is currently not providing any graphical modeling language, and it entire scope remains on the semantic representation of process models. This example depicted in figure 5 shows the nature of the building blocks for the process model. *Transformation Development* process in our example has only one activity, named as *Transformation* Activity.

Transformation activity defines its contracts for all the activity interactions with its contents and context. Each contract for *Transformation* activity shown in the figure is represented by a black and a white block. The black block is representing the external contract, whereas the white block is representing the internal contract. For external contracts, this figure draws the required contracts at the left hand side of the activity components, whereas the provided contracts are at the right hand side. The placement of internal contracts are opposite of that of external components. Thus *Transformation* activity has defined four external contracts (marked as black blocks), where *Input Metamodel*, *Output Metamodel* and *Input Model* are the required contracts and *Output Model* is a provided contract. Each of these external contracts have corresponding internal contracts (marked as white blocks) with opposite nature, thus *Transformation* activity has four internal contracts, where *Input Metamodel*, *Output Metamodel* and *Input Model* are the provided contracts and *Output Model* is a required contract.

Transformation activity is composed of *Transformation* process, which in turn has four activities as *Pattern Identification*, *Technology Decision*, *Rule Generation* and *Transformation Execution*. A dependency can be witnessed between *Pattern Identification* and *Transformation* activities for *Input Metamodel* and *Output Metamodel*. A straight black line between the internal provided contract *Input Metamodel* of *Transformation* activity and the external required contract *Input Metamodel* of *Pattern Identification* activity, represents the dependency between them. In the General Process Metamodel, a contract specifies the artifact that an activity requires or produces, thus each contract shown in this example has an associated artifact, which is quite comprehensible from the name encoding of the contracts. As discussed earlier, figure 5 is a representational model, so it does not show the assigned roles for each activity. In case of *Transformation* activity, a system analyst is the *performer* and is assisted by *tools* like Validation tool and Transformation tool.

Pattern Identification activity is responsible for identifying the matching patterns among input metamodel and output metamodel. This activity produces a pattern list that is made available to its context through the *Matching Pattern List* contract. *Technology Decision* activity is dependant on *Pattern Identification* for the *Matching Pattern List* and on *Transformation* activity for *Input Metamodel* and *Output Metamodel*. Once the technology is chosen for the transformation, the rules are generated for the transformation through the *Rule Generation* activity. *Rule Generation* activity produces a *Transformation Definition* for carrying out the transformation on the input model to generate the output model. *Transformation Execution* activity takes the *Transformation Definition* from *Rule Generation* and *Input Metamodel*, *Output Metamodel* and *Input Model* from *Transformation* activity through its specified required contracts. This in turn produces the *Output Model* for which *Transformation* activity is dependent on *Transformation Execution*. Finally, *Transformation* activity specifies the provided external contract *Output Model* for its context.

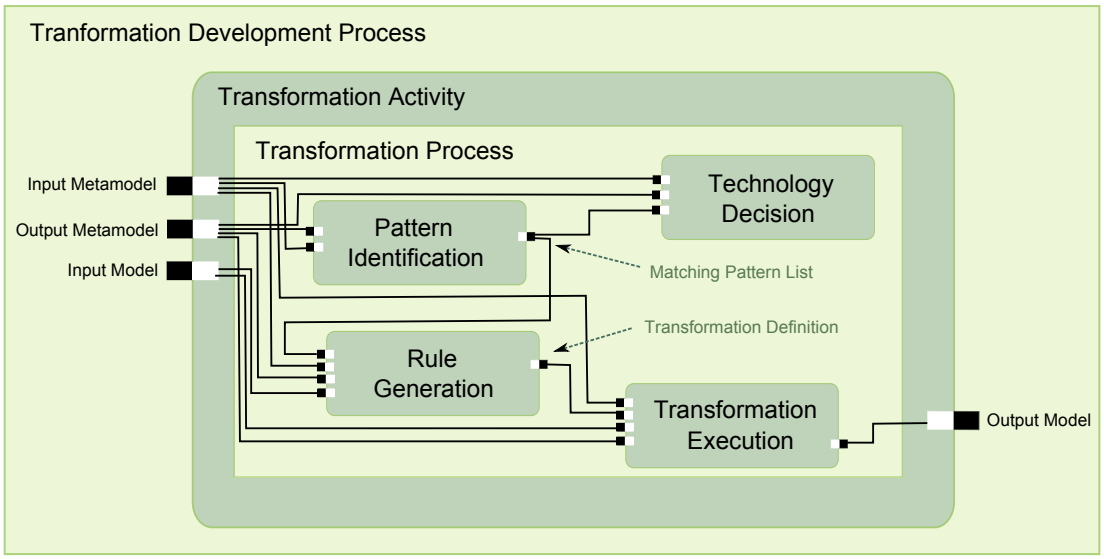

Fig. 5. Contract Package

Out of the activities defined in the *Transformation Development* process, all the activities are semi-automatic activities except *Technology Decision* which is a manual activity that does not need any intervention of tools. *Transformation* activity is a composite activity that relies on the *Transformation* process for its implementation. All the other activities in the example are primitive activities, as they do not compose any process and have a procedural implementation definition. This representational figure does not specify the pre-conditions on the artifacts, which are added in the implementation details of these activities. The dependencies between the activities are realized by using an event based mechanism in our framework.

VI. Conclusion

We have presented a process modeling approach that models activities as components which have their defined contracts. The overall hierarchical structure of the process metamodel allows to model processes at different abstraction levels. Moreover, the complete framework is aimed to be defined using three metamodels i.e. General, Application and Instance metamodels, which helps capture the semantics of processes at all development stages. By accepting models as required and provided artifacts and extending support for model transformations, processes are adapted with the capabilities to support Model Driven Engineering. Moreover, the inspirations from component based architecture add to the expressiveness of software process modeling by using a familiar jargon for the domain experts.

References

[1] D. Ardagna, C. Ghezzi, and R. Mirandola, "Rethinking the use of models in Software Architecture," in *Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures*, ser. QoSA '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1–27.

[2] R. Van Der Straeten, T. Mens, and S. Van Baelen, "Challenges in Model-Driven Software Engineering," in *Models in Software Engineering*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, vol. 5421, pp. 35–47.

[3] A. Forward and T. C. Lethbridge, "Problems and opportunities for Model-centric versus Code-centric Software Development: A survey of software professionals," in *Proceedings of the 2008 international workshop on Models in software engineering*, ser. MiSE '08. New York, NY, USA: ACM, 2008, pp. 27–32.

[4] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, "Business process management: A survey," in *Business Process Management*, ser. Lecture Notes in Computer Science, W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, Eds., vol. 2678. Springer, 2003, pp. 1–12.

[5] OMG, "Software and Systems Process Engineering Metamodel Specification," Version 2.0, April 2008. [Online]. Available: http://www.omg.org/spec/SPEM/2.0/

[6] E. D. Nitto, L. Lavazza, M. Schiavoni, E. Tracanella, and M. Trombetta, "Deriving executable process descriptions from UML," in *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*. New York, NY, USA: ACM, 2002, pp. 155–165.

[7] S. C. Chou, "A Process Modeling Language consisting of high level UML-based diagrams and low level Process Language," *Journal of Object Technology*, vol. 1, no. 4, pp. 137–163, Sep. 2002.

[8] A. Koudri and J. Champeau, "MODAL: A SPEM Extension to improve Co-design Process Models," in *New Modeling Concepts for Today's Software Processes*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6195, pp. 248–259.

[9] R. Bendraou, M.-P. Gervais, and X. Blanc, "UML4SPM: An Executable Software Process Modeling Language providing high-level abstractions," *Enterprise Distributed Object Computing Conference, IEEE International*, vol. 0, pp. 297–306, 2006.

[10] R. Bendraou, B. Combemale, X. Crégut, and M.-P. Gervais, "Definition of an eXecutable SPEM2.0," in *14th Asian-Pacific Software Engineering Conference (APSEC)*. Nagoya, Japan: IEEE Computer Society, Dec. 2007, pp. 390–397.

[11] V. Englebert and P. Heymans, "Towards more extensible metacase tools," in *Advanced Information Systems Engineering*, ser. Lecture Notes in Computer Science, J. Krogstie, A. Opdahl, and G. Sindre, Eds. Springer Berlin / Heidelberg, 2007, vol. 4495, pp. 454–468.