First Workshop on Process-based approaches for Model-Driven Engineering (PMDE)
June 7, 2011
Birmingham, UK

Proceedings

In conjunction with
ECMFA 2011 conference,
June 6-9, 2011, Birmingham, UK

Editor: Laboratoire IRIT, http://www.irit.fr **ISBN : 978-2-917490-19-8**

Contents

Foreword

Welcome to the first edition of the Process-centered approaches for Model-Driven Engineering (PMDE), held in Birmingham, UK, on June 7th, 2011.

The PMDE Workshop aims to gather researchers and industrial practitioners working in the field of Model- Based Engineering, and more particularly on the use of processes to improve software reliability and productivity. Indeed, despite the benefits brought by the Model-Driven Engineering approach, the complexity of today's applications is still hard to master. Building complex and trustworthy software systems in the shortest time-to- market remains the challenging objective that competitive companies are facing constantly. A more challenging objective for these companies is to be able to formalize their development processes in order to analyze them, to simulate and execute them, and to reason about their possible improvement.

The PMDE workshop's goal in this edition is to present research results or work-in-progress in all areas of process-based approaches for model-driven engineering. The main topics that where targeted this year by accepted papers range from collaborative processes, process enactment, DSLs to support MDE development, to the proposition of new process modeling techniques and frameworks.

The organization of this first edition of the PMDE workshop would not have been possible without the dedication and professional work of many colleagues. We wish to express our gratitude to all contributors who submitted papers. Their work formed the basis for the success of this year's edition and we hope, for the upcoming editions of PMDE. We would also like to thank the Program Committee members and reviewers for volunteering their time to help assess the submissions and guarantee the quality of the workshop.
Finally, we are also grateful to the ECMFA 2011 organizing Committee, particularly to Behzad Bordbar and Rami Bahsoon for their help and valuable advices.

June, 2011, The Organizing Committee.

Reda Bendraou
Redouane Lbath
Marie-Pierre Gervais
Bernard Coulette

Organization

Program committee

Colin Atkinson (University of Mannheim, Mannheim, Germany)

Behzad Bordbar (University of Birmingham, UK)

Jacky Estublier (University of Grenoble, France)

Christian Haerdt (EADS, Germany)

Jason Xabier Mansell (TECNALIA - ICT/European Software Institute, Spain)

Larrucea Uriarte Xabier (TECNALIA - ICT/European Software Institute, Spain)

Leon J. Osterweil (University of Massachusetts, USA)

Richard Paige (University of York, UK)

Garousi Vahid (University of Calgary Alberta, Canada)

Organizers

Bendraou Reda (LIP6, France)

Lbath Redouane (IRIT, France)

Coulette Bernard (IRIT, France)

Gervais Marie-Pierre (LIP6, France)

Collaborative Processes in the Real World: Embracing their Ad-hoc Nature

Komlan Akpédjé KEDJI[1], Bernard COULETTE[1], Mahmoud NASSAR[2], Redouane LBATH[1], and Minh Tu TON THAT[3]

[1] IRIT, Toulouse, France,
{kedji, bernard.coulette, redouane.lbath}@irit.fr
[2] ENSIAS, SIME, Rabat, Morocco nassar@ensias.ma
[3] University of Toulouse, France minhtutonthat@gmail.com

Abstract. In real world projects, collaboration in not only omnipresent, but efficient collaborative processes are also hard to get right, especially when the cohesion needed is high. In software engineering, a sizable amount of information needed to understand and support collaboration is unavailable before the start of the project. We argue that such lack of information is essential to the field, and needs to be embraced, rather that fought, or worse, ignored. We believe collaboration in the real world to be a feed-back process, where short-term decisions guide collaboration, and where the way collaboration unfolds in the project, in turn, guides future decisions about collaboration. Practitioners should be given the ability to express such evolving understanding about collaboration in a formalism suited for easy representation and tool-provided assistance. To this end, we propose an extension to the Software & System Process Engineering Meta-Model (SPEM), which introduces concepts needed to represent precise and dynamic collaboration setups that practitioners create to address ever-changing challenges.

Keywords: process design; process planning; collaboration; collaborative work

1 Introduction

Collaboration can be simply defined as the act of working together, towards a common goal, and is a pervasive concern in any collective endeavor. While process formalisms and standards like SPEM [11] exist, support for the description of ad-hoc collaboration (that is, collaboration that is done because it happens to be needed, and not because it was planned) is lacking. We show that in the field of computer engineering, collaboration is usually ad-hoc, and that this is not by accident, but rather by necessity. We embrace that necessity in our quest to build a better mental model of collaboration, so as to be able to offer effective tool support to practitioners. Our approach naturally results in a feedback loop between reality and the model, where decisions taken in the project are continuously injected in the model, and at any given moment, the state of the model guides future decisions.

We take the MDE (Model-Driven Engineering) approach, and build a metamodel suited to the description of ad-hoc collaboration practices, which extends SPEM. This paper shows, on an example taken from a real world project, how appropriate our conceptualization is for actual software projects. It should be noted that our effort is not restricted to face to face or synchronous interactions, but covers any collective work that escapes the plan-and-execute scheme.

The rest of this paper is organized as follows. Section 2 characterizes ad-hoc collaboration to build an understanding which we formalize in Section 3 as a metamodel. Section 4 describes and represents a real world collaboration scenario with the proposed concepts. Section 5 summarizes some related works.

2 Ad-hoc Collaborative Processes

Collaboration happens whenever two or more individuals work on the same task or interact with the same work product. Collaboration has been defined as "a collective effort to construct a shared understanding of a problem and its solution" [16].

Various concerns such as power sharing, responsibility assignment, appropriate recognition, giving credit, coordinating actions, trust, etc., can easily undermine collaborative efforts, by incurring to much cognitive overhead on participants, and taking the focus away from their actual tasks. In software engineering, these problems are further complicated by worldwide distribution of teams, the amount of knowledge sharing required, communication among people with different expertise, and high complexity and interdependency of artifacts [9].

Thus, people usually collaborate only when the need arises. While there can be some prior planning, collaboration, especially in software engineering, is mostly ad-hoc [3, 15]. Whereas planned collaboration happens because it was on the agenda, ad-hoc collaboration is prompted by a new issue or new considerations in an existing issue. A field study found ad-hoc collaboration to amount to as much as 69% of all collaborative work in software development [15].

Any tool that seeks to support collaboration had better speak the language of its users, so as to not increase the cognitive overhead they are already fighting with, and provide a frictionless surface for development [2]. When ad-hoc collaboration occurs in software engineering, the concepts involved for the practitioners are the actual people doing the job, what each of them is doing, and which artifacts they are manipulating.

However, the central concepts used in process models like SPEM (role, product, task) are at a much higher level of abstraction than those used by practitioners to understand collaboration, thus hiding needed information:

- Role. How do we describe interactions between people playing the same role, but still doing different things like coding two dependant components, when the very need for two people surfaces only in the middle of the project?
- Tasks. When we find out that more than one person is needed for a task (which is the very definition of collaboration), how can we, for example,

specify work dependency, when we lack a concept to represent work done by a specific person in the context of a task?
- Products. When a group of participants have each a copy of a work product in their workspace, how do we describe their agreement for merging changes to produce an official version of the work product, while lacking the ability to identify each of those copies?

SPEM is glaringly silent on such issues, and even proposes that these issues be solved by project manager as they see fit, when instantiating process models[4]. We need to frame collaboration in terms of more appropriate concepts to effectively support the kind of ad-hoc collaboration that is dominant in software engineering.

If we focus on ad-hoc collaboration, and structure it with the concepts familiar to practitioners, it follows that we can't always describe collaboration in a generic way, and just reuse that description for any project. But this is not a limitation, as collaboration practices are ad-hoc precisely because information needed is not available earlier. Postponing the description of ad-hoc collaboration is therefore better suited to taking into account whatever creative collaboration setup practitioners create to answer their actual needs [6].

3 The CM_SPEM Metamodel

Our conceptualization is presented as a metamodel, CM_SPEM (Collaborative Model-based Software & Systems Process Engineering Metamodel[5]), that extends SPEM [11] but adding packages. Due to space limitation, the metamodel described in this article focuses on the structural description of collaboration, and the OCL (Object Constraint Language) [13] rules which specify the static semantics of the metamodel as well-formedness rules have also been left out. A more detailed overview is available in [4].

In the current work, we chose to describe collaboration using the same central concepts which structure ad-hoc collaboration: Actor, ActorSpecificTask, and ActorSpecificArtifact. These concepts can be seen as refinements of the concepts we borrow from SPEM: RoleUse (from SPEM::ProcessStructure), TaskUse (from SPEM::ProcessWithMethods), and WorkProductUse (from SPEM::ProcessStructure). We encode knowledge about collaboration setups as relationships between the previous concepts, and use the SPEM Kind mechanism to allow them to have user-defined qualifications. This helps avoid the temptation to provide a fixed set of relationships which cannot cover all collaboration setups, while still providing some general structure. These structural concepts are grouped in the CM_SPEM::CollaborationStructure package, which reuses SPEM::ProcessStructure and SPEM::ProcessWithMethods.

[4] See Section 16 "Enacting SPEM 2.0 Processes" of the standard [11].

[5] This work is part of a project which focuses on the MDE (Model Driven Engineering) approach to software development, hence the name of the metamodel. However, this paper is restricted to the collaborative features of the metamodel.

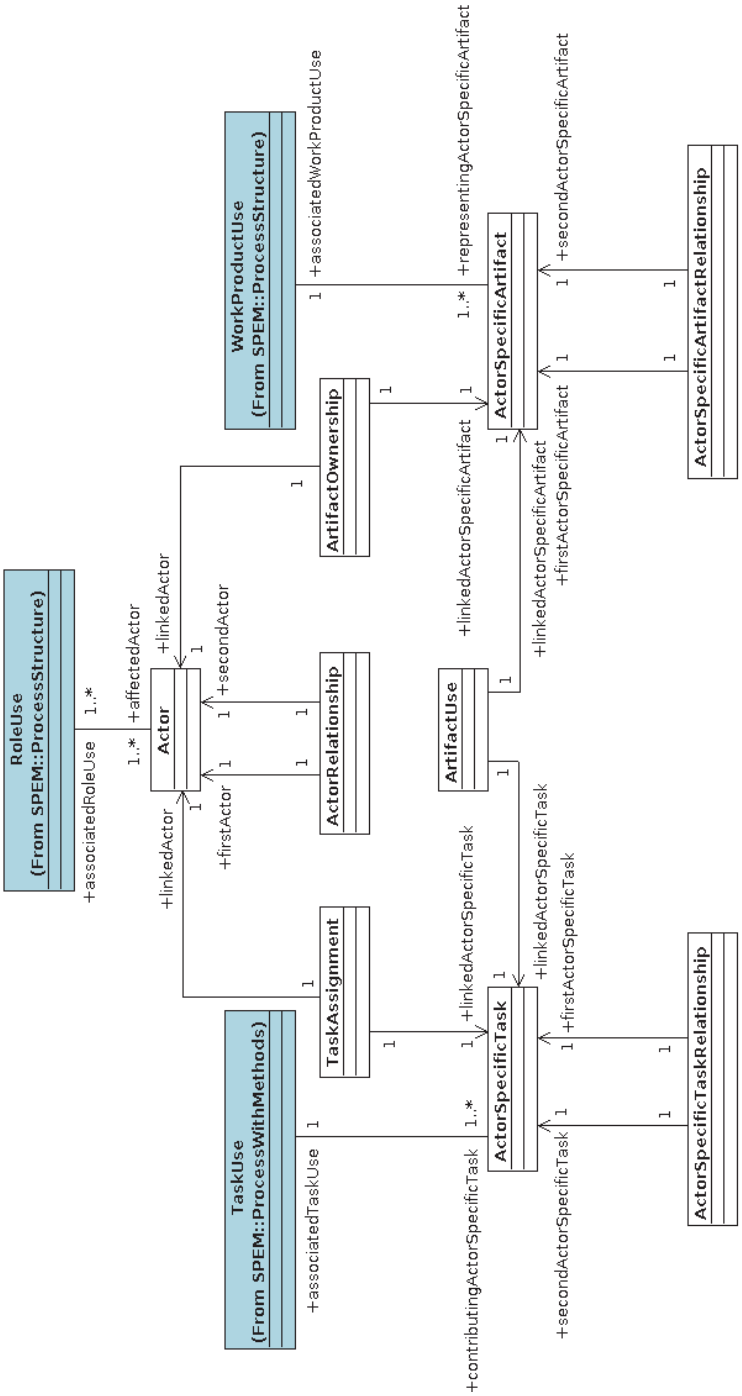


Fig. 1: Overview of the Collaboration Structure package. Concepts in blue are borrowed from SPEM.

3.1 Central Concepts

The three central concepts in CM_SPEM are Actor, ActorSpecificTask, and ActorSpecificArtifact. All of them extend ExtensibleElement (from SPEM::Core).

An actor identifies a specific human participant in a project. It is linked to instances of RoleUse (SPEM::ProcessStructure) to specify that the actor plays the corresponding roles. An ActorSpecificTask is a unit of work done by a specific actor, towards the execution of a TaskUse. An ActorSpecificArtifact[6] is a physical occurrence of a WorkProductUse, in the personal workspace of a specific actor. This is the personal copy of the actor, and is manipulated only by him/her[7].

3.2 Relationships

All the defined relationships are subclasses of SPEM::Core::ExtensibleElement (which allow them to have user-defined qualifications) and SPEM::Core::BreakdownElement (which allows them to be nested by SPEM containers like Activity).

A first set of relationships is used to associate instances of different concepts: TaskAssignment links an Actor to an ActorSpecificTask assigned to him/her, ArtifactOwnership relates an Actor to an ActorSpecificArtifact in his/her workspace, ArtifactUse relates an ActorSpecificTask to an ActorSpecificArtifact manipulated when carrying it out. This set of relationships can be conceived as a mirror of the set of links defined in SPEM between RoleUse and Activity (ProcessPerformer), RoleUse and WorkProductUse (ProcessResponsibilityAssignment), and, Activity and WorkProductUse (ProcessParameter)[8].

The second set of relationships is used to relate instances of the same concept:

- ActorRelationship, which is used to describe a couple of actors. An ActorRelationship can be used to state that an actor A reports to actor B in a certain task, so that all contributions made by A, in the context of that task, are by default sent to B.
- ActorSpecificTaskRelationship, which is used to describe a couple of ActorSpecificTask. An ActorSpecificTaskRelationship can for example specify temporal dependencies between the works done by two different actors in the context of the same TaskUse.

[6] In SPEM, Artifact (a tangible work product) is one of the three work product kinds. The two other are Outcome (a non-tangible work product) and Deliverable (a group of artifacts). While "Artifact" in ActorSpecificArtifact has a wider semantic (anything that represents a WorkProduct), the similarity with "Artifact" as used in SPEM is defensible. Indeed, an Outcome usually does not have distinct representations for each participant, and a Deliverable can be considered as a special Artifact.

[7] Arguably, there could be a lot of such copies, for the same actor, in different version control branches for example. However, we are only interested in the fact that any one of these artifacts is only manipulated by a single actor

[8] See Figure 9.4 of SPEM 2.0 [11]

– ActorSpecificArtifactRelationship, which is used to describe a couple of ActorSpecificArtifact. An ActorSpecificArtifactRelationship can for example specify that an artifact A is the reference version of an artifact B (A and B represent the same work product).

4 Ad-hoc Collaboration in a Real World Project

This section describes an ad-hoc collaboration scenario extracted from a real world project, using the concepts introduced in the previous section.

4.1 Context

Mozilla is a project which produced a number of well-known software, among which the BugZilla[10] bug-tracking software, and the Firefox web-browser. BugZilla is used to track bugs in the Firefox project.

We are interested on how different people collaborate inside the Mozilla project to solve a single bug in Firefox' source code, using BugZilla. We argue that with our conceptualization of collaboration, some of the functionality provided by an application like BugZilla can be simply implemented, using information from CM_SPEM process models. This is a major progress, because many other support tools (like continuous integration servers, technical wikis, automatic development environment configuration scripts, etc.) could be built using information from the same CM_SPEM process model.

Fixing a bug can be considered as an activity, where roles like "Developer", "User", "Quality manager" are involved, and where the artifacts impacted by the bug are manipulated. The general lifecycle of bugs[9] in Bugzilla is shown in Figure 2. A bug is usually reported by an end user, and starts in the state "UNCONFIRMED". When a developer successfully reproduces the bug, it moves to the "CONFIRMED" state. The bug can them be assigned to a developer who will develop a fix, while the bug is in the state "ASSIGNED" or "IN PROGRESS". Eventually, the developer finds a fix, and the bug is marked as "RESOLVED", and moves to the state "VERIFIED" when someone from the quality assurance team confirms that the problem has been actually solved (with no regression) by the provided fix.

For any particular bug, a custom collaboration setup can be created, to account for its peculiarities. As an illustration, we will study bug number 461304, titled "Loading URL by pressing ENTER on already present URL in location bar doesn't maintain URL encoding" [10] . Simply put, this bug is a character-set encoding issue when selecting an entry in the drop-down menu that the browser shows when typing web URLs in the address bar.

[9] http://www.bugzilla.org/docs/tip/en/html/lifecycle.html
[10] https://bugzilla.mozilla.org/show_bug.cgi?id=461304

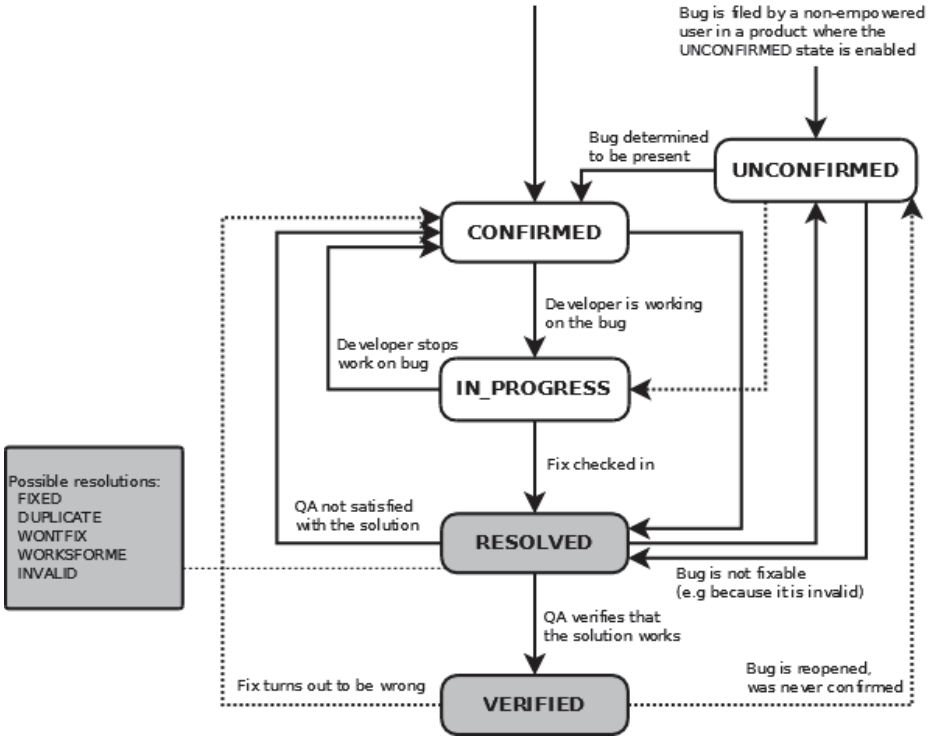


Fig. 2: The lifecycle of a bug in BugZilla (Source: BugZilla Documentation [10])

4.2 Informal Description

Bug 461304 has been reported Kai de Leeuw, in October 2010. José Jeria confirmed the bug, and added this was a windows-only issue. Mike Beltzner, a release driver at Mozilla, flagged the bug as "blocking-firefox3.5-", which means that someone proposed that this particular bug be resolved before Firefox 3.5 ships, and the release driver decided it should not block the release. Another release driver, Dietrich Ayala, flagged it as "wanted-firefox3.6+" which means he wants that bug to be resolved before Firefox 3.6 is released (by this will not block the release if it is not). Mike Beltzner then explicitly asked Gavin and Madark to localize the bug. Gavin found a conflict between the system encoding in windows (Windows-1252), and the encoding of natively used for URLs in Firefox (UTF-8).

While investigating, Mike Beltzner suggested that this bug be marked a duplicate of bug 333859. But it became clear later that fixing bug 333859 will have too far reaching consequences (compatibility breaking). Another bug, 393246, was identified, as another duplicate, and fixing it will have less broad consequences. So it was decided to fix bug 393246, which will also fix the original bug 461304.

All the afore mentioned people (and many others) were added to the "CC list" of the bug, meaning that they get emailed whenever a change is made to the bug in BugZilla.

4.3 Description Using CM_SPEM Concepts

There are a number of actors involved:

- Kai de Leeuw is a user
- José Jeria, Gavin, and Madark are developers
- Mike Beltzner and Dietrich Ayala are release drivers

A number of actor specific tasks can be identified:

- "Report Bug", carried out by the user Kai de Leew. Note that in this particular case, the task is identified, then affected to someone who carries it out. The user just stumbles upon the bug during normal browser usage, and decides to report it. Thus, this actor specific task in the model merely records something that has been done. This demonstrates how some tasks in a project cannot explicitly appear in a process model which exclusively focuses on planning.
- "Confirm bug", carried out by developer José Jeria.
- "Identify bug cause", carried out by Gavin
- "Fix bug", which will be carried out by the developer assigned to the bug (as of today, no one has been assigned to this bug).
- "Confirm fix" which will be carried out by a member of the QA (Quality Assurance) team. Note that this actor is not yet identified, as bugs are usually fixed before a QA team member (depending on his workload and his area of expertise) reviews it for fix confirmation.

The conversation happening around the bug, depending on how it is recorded, can be considered as an artifact on its own. In the case of BugZilla, the conversation is stored in BugZilla's database, outside of the actual repository. Thus, we focus only on the artifacts impacted by the bug:

- The file[11] which tracks user preferences (URL encoding can be specified as a preference in Firefox).
- The file which handles URL encoding in Firefox[12]

Each copy of these files in a developer directory is an actor specific artifact on its own, which can evolve independently of others. Typically, the copy in the repository of the developer assigned to the bug will evolve first, and the changes made will be sent to an integrator (not shown in the actor list for simplicity).

Some CM_SPEM relations can be identified as well in this situation:

[11] `seamonkey/source/modules/libpref/src/init/all.js`

[12] `docshell/base/nsDefaultURIFixup.cpp`

– The actor specific task “Identify bug cause” can be carried out only when “Confirm bug” is complete (in other words, no developer is assigned to work on a bug before it has been reproduced by the Firefox team). This is an ActorSpecificTaskRelationship.
– The user that reported the bug can provide feedback on the fixes proposed by the developer assigned to the bug (practically, it means every time the developer introduces a patch for that bug, the user can test it, and report results). This is an ActorRelationship.
– In this particular case, as another bug was marked as a duplicate of the original bug, yet another ActorRelationship holds between the assignee of the original bug and the assignee of the duplicate bug. The latter, after fixing the bug assigned to him, must inform the former (so that the orginal bug can also be marked as resolved). This particular relationship is already implemented by BugZilla.

The above relations can materialize in a number of ways. For example, BugZilla has a feature called the “CC list”, which contains the email addresses of people who must be notified by email each time there is a significant change to a bug (like a change of state). The content of the CC list can be automatically derived from the above relations (by deriving all involved actors from the relationships, and then retrieving their email addresses).

4.4 Other Exploitation Possibilities

In this section, we showed how information stored into CM_SPEM models can be used to provide a BugZilla-like functionality. Other possible usage scenarios include:

– Automatically configure development environments (editors, version control systems, …).
– Extract visualizations which helps practitioners understand what is going on. Example include displaying how connected people are by their work (which helps improve the organization of the team), or how much work load a person has at a given moment. Field studies in software engineering can also reuse this data.
– Provide notifications. Relationships can be used to send useful information to the people who need it (the “CC list” in the case of BugZilla is an example). While information is desirable, practitioners can easily become drowned in irrelevant information. The notifications sent can be filtered by considering the semantics of the relationships each participant is involved in.
– Inform people of what their peers are actually currently doing, so they can know when they can interrupt them, and when they had better wait.

5 Related Works

Previous contributions of interest to our work relate to flexible instantiation of process models, extending process metamodels like SPEM to allow a more pre-

cise definition of process models, and giving practitioners the ability to enhance process models with their evolving understanding of what they are doing.

Killisperger et al., developed a framework for flexible process instantiation [8]. The goal is to assist step-by-step tailoring and instantiation of generic and complex process models. While the progressive approach is shared by our proposal, they focus on constraining tailoring and instantiation (syntactic and organizational constraints), while we focus on supporting collaboration.

There have been other efforts to extend SPEM. For example, [14] defines a formalism based on petri nets geared towards precise definition of MDE process models and process execution tracking. It however ignores resource allocation and roles definition, and concentrates on process steps (described with links, models, flow, resource, etc.) This formalism is more detailed than SPEM, but follows the same "define and execute" approach.

Witshel et al. [17] identified the need to make business process execution more flexible. They start from the insight that – while offering valuable context information – traditional business process modeling approaches are too rigid and inflexible to capture the actual way processes are executed. This is also the case in software engineering, where the execution is even less predictable. The paper explores how practitioner knowledge can be leveraged to enhance BPMN [12] process models, by allowing workers to leave semi-structured comments on BPMN activities. While this functionality is not natively present in our metamodel, it could be leveraged by considering our concepts (actor, actor specific task, actor specific artifact) and their relations as a way of annotating activities with the details of their collaborative execution.

The ISO/IEC 24744 standard (Software Engineering – Metamodel for Development Methodologies) [7] is a metamodel which addresses some of the rigidity of SPEM. Its levels, metamodel, method, and endeavour, roughly correspond to the metamodel, model, and process instance levels in SPEM. However, ISO/IEC 24744 has the ability to represent a concept defined at the metamodel level directly at the endeavour level (SPEM's process instance level) and replaces rigid "instance-of" relations with representation relations. This has been shown[5] to be better suited to agile development and progressive decision-making in software projects, which makes our contribution on the ad-hoc nature of collaboration similar to the ISO/IEC 24744 effort. Besides the fact that ISO/IEC does not explicitly address the issue of collaboration, the choice of SPEM as a starting point has been motivated by project context (SPEM closely follows the model driven engineering approach, which is a central theme in the wider context of this work).

In [6], Grudin et al. proposed a methodology for the design of collaborative environment which is halfway between the top-down approach of put-all-the-knowledge-in-at-the-beginning and the bottom up approach of just-provide-an-empty-framework. Witshel et al. [17] used this approach to design a "task pattern" formalism (task patterns are defined as "abstractions of tasks that provide information and experience that is generally relevant for the task execution. By abstraction we mean common features of a family of similar tasks, which aim

at the same goals under similar conditions"). Task patterns are instantiated by assigning real persons to the positions defined in the task pattern (concerned people can refuse or accept). A user can enhance a task pattern while executing it. The enhancements are stored locally, reused automatically for subsequent instances of the task pattern, and can be published so others can use it.

In [1], Alegria et al. use three perspectives (views: role, task, work product) to visualize process models, and discover their defaults. This visualization approach could be exploited in our proposition by applying it to actors, actor specific task, and actor specific artifacts, so as to provide a precise and hopefully insightful view into the actual execution of a process model.

6 Conclusion

This work introduced CM_SPEM, a SPEM extension for the description of ad-hoc collaboration work. Our guiding principle is that collaboration should be conceptualized using the ideas most familiar to people collaborating. This is reflected in our choice of concepts and the way they are combined.

We showed, on a real world example, that information on collaboration usually becomes gradually available, which makes it impossible to provide all the details of collaborative process models at the start of project. Instead, we advocate a feedback loop between how collaboration unfolds in reality and what is (initially) recommended by the model.

While enriching process models is useful, this could become time-consuming for practitioners. However, adding collaboration information to process models need not be done by hand, but could be partially automated using the CM_SPEM API. For example, ActorRelationships could be identified by analysing version control history, and their addition suggested to practitioners.

A graphical notation for CM_SPEM models, and a visual editor based on that notation have been developed, but are not presented here due to insufficient space. The editor is based on Eclipse, and provides some consistency checks when editing CM_SPEM models. There is also a project plan generator for Microsoft Project and GanttProject in the making. We are currently formalizing the behavioral aspects of ad-hoc collaboration as an extension of CM_SPEM.

Other future directions include the definition of a complete query API for CM_SPEM process models (used by development support tools to extract information from CM_SPEM models), and develop an ecosystem of tools which materialize the ultimate goal of our metamodel.

Acknowledgment

This effort is part of the Galaxy Project, funded by ANR – France.

References

1. Alegría, J., Lagos, A., Bergel, A., Bastarrica, M.: Software Process Model Blueprints. New Modeling Concepts for Today's Software Processes pp. 273–284 (2010)
2. Booch, G., Brown, A.: Collaborative development environments. Advances in Computers 59, 1–27 (2003)
3. Cherry, S., Robillard, P.: The social side of software engineering–A real ad hoc collaboration network. International journal of human-computer studies 66(7), 495–505 (2008)
4. Coulette, B., Kedji, K., Nassar, M.: Vers un métamodèle pour la représentation des procédés idm collaboratifs. Proc. of 7ièmes Journées sur l'Ingénierie Dirigée par les Modèles (June 2011)
5. Gonzalez-Perez, C.: Supporting situational method engineering with iso/iec 24744 and the work product pool approach. Situational Method Engineering: Fundamentals and Experiences pp. 7–18 (2007)
6. Grudin, J., McCall, R., Ostwald, J., Shipman, F.: Seeding, Evolutionary Growth, and Reseeding: The Incremental Development of Collaborative Design Environments. Coordination theory and collaboration technology p. 447 (2001)
7. Henderson-Sellers, B., Gonzalez-Perez, C.: Standardizing methodology metamodelling and notation: An ISO exemplar. Information Systems and e-Business Technologies pp. 1–12 (2008)
8. Killisperger, P., Stumptner, M., Peters, G., Grossmann, G., Stückl, T.: A Framework for the Flexible Instantiation of Large Scale Software Process Tailoring. New Modeling Concepts for Today's Software Processes pp. 100–111 (2010)
9. Mistrík, I., Grundy, J., van der Hoek, A., Whitehead, J.: Collaborative Software Engineering: Challenges and Prospects. Collaborative Software Engineering p. 389 (2010)
10. Mozilla: The bugzilla project. http://www.bugzilla.org/ (2011)
11. OMG: Software process engineering metamodel, version 2.0. http://www.omg.org/spec/SPEM/2.0/ (2007)
12. OMG: Business process model and notation, version 1.2. http://www.omg.org/spec/BPMN/1.2/ (2009)
13. OMG: Object constraint language. http://www.omg.org/spec/OCL/2.2/ (2010)
14. Porres, I., Valiente, M.: Process definition and project tracking in model driven engineering. Product-Focused Software Process Improvement pp. 127–141 (2006)
15. Robillard, P., Robillard, M.: Types of collaborative work in software engineering. Journal of Systems and Software 53(3), 219–224 (2000)
16. Roschelle, J., Teasley, S.: The construction of shared knowledge in collaborative problem solving. NATO ASI Series F Computer and Systems Sciences 128, 69–69 (1994)
17. Witschel, H., Hu, B., Riss, U., Thönssen, B., Brun, R., Martin, A., Hinkelmann, K.: A Collaborative Approach to Maturing Process-Related Knowledge. Business Process Management pp. 343–358 (2010)

A Framework for Integrating ESL Tools

Ali Koudri[1], Teodora Petrisor[1], Joel Champeau[2], and Vincent Leilde[2]

[1] Thales Research and Technology
[2] ENSTA Bretagne STIC/IDM

Abstract. Today embedded systems are complex and their development is based on the use of a large set of tools. Although point to point tool integration can be efficient, it is not very well adapted to changing because it remains costly and human expensive. To solve this problem, several approaches on tool integration have been proposed. Even if those approaches handle the flexibility and the obsolescence issues, the tool synergy cannot still be achieved due to major problems such as technological or semantic gaps. Recently, new techniques such as model based engineering and communities such as OSLC have emerged. They represent a promising approach for significant improvement of integration frameworks. In this paper, we present the basics for an integration framework supporting HW/SW co-design development that takes benefits from modeling and current standards. This work is part of the iFEST ARTEMIS project.

1 Introduction

Today, development of real-time embedded systems is more and more stressed by higher demands in terms of functionalities as well as economical factors pushing the competitiveness by reducing costs and time to market. Indeed, developing, producing and maintaining embedded systems raise a number of challenges since they are inherently heterogeneous (mixing software and hardware parts, analog and digital parts, etc.). The development of embedded systems constitutes actually a multidisciplinary endeavor as it involves the cooperation of a large number of experts from various domains over the various stage of the development (Requirements Engineering and Analysis, Design and Implementation, Verification and Validation, Lifecycle Management). As such, stakeholders of the process have to face issues related to external communication (customer) as well as internal communication (between the various experts involved in the development). Those issues have a really strong impact on the cost, the time-to-market and the quality of the delivered products.

In this context, we think that having more fluid communication between the various stakeholders would help to improve the overall productivity and the quality of the products. To achieve this goal, we shall consider the two most important communication styles that occur in any development project: the oral communication and the communication relying on the exchange of artifacts produced by the various tools used during the process lifecycle (MS-Word, MS-Excel, Doors, GNU C Compiler, etc.). The first one is addressed by the natural

language processing and is out of the scope of this paper. Rather, we address in this paper the improvement of the communication between the various kinds of tools generally used in a co-design process targeting the development of real-time embedded systems. By communication, we mean "intelligent communication" that takes into account not only syntactic considerations but also semantic ones as well as lifecycle issues. We present in this paper the basics for a framework dedicated to integrating ESL (Electronic System Level) tools which aims not only to reduce costs and time-to-market, but also to improve the quality of the products through early defects detection and correction. This work is done in the context of the iFEST ARTEMIS project [8].

This paper is organized as follow: the second section presents the background and the motivations of our work; the third section presents the basis for the integration framework platform; the fourth section discusses the expected benefits of the integration framework platform; finally, the conclusion draws the current state of our work and its perspectives.

2 Background and Motivation

In nowadays practices, tools supporting the development of information systems are not very well integrated. This is particularly the case for HW/SW co-design, which often addresses complex issues related to multi-cores and parallelism, and where tools are still sparse and not very mature. An appropriate tool chain would allow engineering assets to be developed and/or reused, and applied for different contexts with different purposes. Indeed, development of real-time embedded systems requires establishing tool chains tailored to fit production characteristics as well as business organization and concerns. Since requirements and supporting tools can evolve during the process lifecycle, such tool chains require to be maintained easily and seamlessly. For example, the necessity to reduce cost can push companies to change a tool by another one supporting the same features during one project. Such change can raise numerous risks related to bugs or learning curves that are potentially costly. This assertion is also right for the introduction of a new version of a given tool. Besides, lifecycle support represents an important issue as re-engineering can occur when hardware components become obsolete or when new software versions are developed and deployed.

In this context, the adoption of a tool integration technology supporting the adding, the removing and the replacement of tools to support a given process is of a great necessity. Tools integration should enable smooth evolution from one generation to the next, with much lower effort than it takes today, retaining the best of previous generations and adding upgrades in a formal environment, minimizing the risks. Moreover, if such integration technology can also support process orchestration, it would greatly improve the quality of the process and thus, the resulting products. Tools integration has been studied by many scientific and technological communities since the 70s [5, 14, 4, 10]. For instance, according to [5, 14], there are various levels for tools integration:

- *Data integration* is concerned with the provision of mechanisms that support the sharing of data among tools and managing the relationships among data objects produced by different tools. This kind of integration deals with traceability, consistency, completeness and the granularity of the information being exchanged,
- Control integration refers to the ability of tools to perform notifications to other tools as well as to run other tools,
- *Presentation integration* is concerned with providing a common "look and feel" for a number of tools from the user's perspective,
- *Process integration* is concerned with the roles of tools in the overall development process,
- *Platform integration* is concerned with basic inter-operability among tools, traditionally referring to system services provided by a middleware or an operating system.

Not all the dimensions have to be considered for a given project, it depends mainly on the organization and the project characteristics. For example, presentation integration may not be fully desirable when different disciplines already have established their ways of presenting information. There can also be some possible dependencies and overlapping between those dimensions. In the following list, we give a brief overview of the main technologies supporting such capability:

- Model-driven engineering includes the use of metamodels and model transformations to support tool integration. This has been a rapidly evolving area for the past 10 years and it represents a promising approach for systematic tool integration [11],
- Process-driven tool integration emphasizes formalized descriptions of processes. Execution of such models allows process orchestration into which the process execution engines performs actions upon tools and artifacts [3],
- Web-services based tool integration uses standardized communication protocols developed in the context of the internet and web for the purpose of tools integration [2],
- Modular tools integration relies on extensible platforms supporting the definition of modular tools as new extensions. For instance, the Eclipse platform [7] complies with such a definition,
- Database and life-cycle management approaches rely on structured information management. They typically use centralized repositories and, sometimes, standardized exchange formats.

While some of the current technologies have a focus on providing support for integrating engineering tools, this is for example true for Modelbus [13], others such as Eclipse or Jazz [6] are more agnostic and provide basic support for integrating any type of tool. Although all those technologies have their pros and their cons, they generally address one or two dimensions among the ones presented above. Then, through this work, we aim to cross a step forward and propose a solution based on a real separation of concerns allowing the seamless integration

of any tool into a co-design tool chain. To achieve this goal, we propose first to characterize any tool used in a co-design according several viewpoints. Later, we aim to support process orchestration based on automated enactment of process models.

In order to reach our objectives, we have defined the following intermediate objectives:

- select, and on very specific occasions develop, interoperable tools for tool chains,
- automate the transformation of tool exchange data when going from one phase in the engineering life cycle to the next one,
- select, and develop as needed, advanced integration technologies which shall be open, tool independent, and preferably standardized,
- automate process steps, i.e. automate one or more development activities.

We think that integration technologies must support both available and future tools, enabling them to be integrated into process flows with minimal disruption and effort. In this context, the iFEST project will emphasize the use of standardized technologies and will promote and support standardization of its integration technologies.

3 Basis for the Integration Framework

As mentioned in the previous section, our integration framework aims to provide means for:

- The development of complex real-time embedded systems requiring the use of heterogeneous and multi-core platforms, according to a well-defined HW/SW co-design process,
- The support of all the development phases from requirements to the production of hardware and software, including the life cycle of the process / product / system. This involves the integration of tools in such a way that the interfaces are tool independent.

We believe it will take a reasonable and straight-forward effort to set up a tool chain supporting each phase of a co-design process. The main reason is that there are plenty of tools available for each phase of a co-design process. Then, our approach to characterize tools used in such process will be pragmatic: we start by analyzing the needs of the different industrial partners involved in the project, representing different domain areas (aerospace, automotive, railways, robotic), and the tool they use for each phase. Based on this idea, we have established a map of used tools for each partner at each main phase of their process (Requirement Engineering and Analysis, Design and Implementation, Verification and Validation, Lifecycle Support).

The figure 1 shows an example of such map for the development of a radar system. We can notice in this figure that a strong emphasis has been put on the life cycle issues. Indeed, the various aspects of the life cycle have considerable

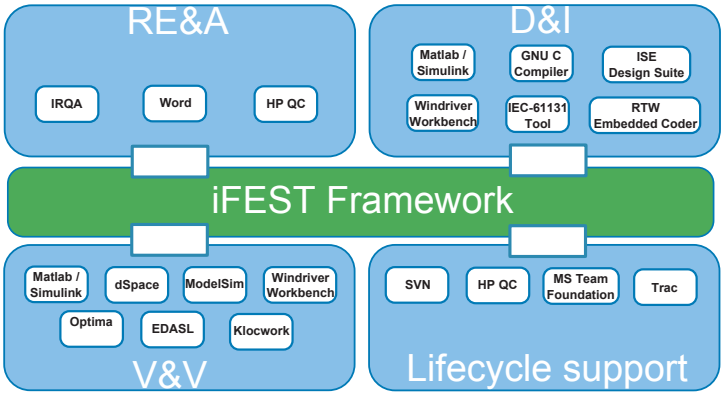


Fig. 1. Co-design Tools for developing a Radar application

impacts not only on the engineering process but also on the properties of the product itself from a structural and architectural point of view. These aspects are costly and difficult to change, once the product has been released, if they have not been considered up-front. This leads to the concept of Design for Life Cycle Support considering issues related to the major phases of development and/or evolution of the product.

Regarding life cycle issues, we aim to achieve advances in the following areas:

- *Design for Changeability* The ability to easily change system components (HW or SW) due to obsolescence, malfunctioning or the introduction of an additional functionality. The changes must be carried out in a cost effective and controlled fashion, i.e. they must not lead to quality or regression issues. The changes are not limited to the product and associated artifacts but they also include individual tools within the tool chains. Since tools are also part of the system environment, they may themselves be subject to obsolescence. Design for Changeability requires tooling support and automation of the change management process to allow rigorous and detailed tracking in terms of change types, reason, impact, ownership, etc.
- *Design for Traceability* Traceability is the degree to which a relationship can be established between two or more artifacts of the product life cycle, especially establishing relationships between predecessors and successors (e.g. requirements and design, design and code, etc). The objective of iFEST is to allow for forward and backward traceability,
- *Design for Modularity* Modularity refers to the degree to which system components can be separated and re-integrated. Design for Modularity aims at minimizing unintended interactions between components, hence pushing towards the decoupling of interfaces. This has major benefits in terms of life cycle support: improving maintenance and quality as well as enabling concurrent engineering shortening time to market.

The real challenge then is to provide a framework enabling the establishment of an HW/SW tool chain in a process centric approach based on the use of models and supporting life cycle management. The tool integration framework will foster architectural design space exploration in a cost effective design while reducing risks related to tools and communication between tools. A major innovation in this respect is the envisioned integration of traditional HW/SW co-design tools (with formalized models and partitioning schemes) and tools that typically belong to the MDE-world (with meta-modeling and transformations). Indeed, the use of model driven engineering will allow us to go from abstract modeling to hardware design and software coding in a cost-effective manner. Meanwhile, our goal is not really about developing new modeling technologies, nor about developing new tools as such. Rather, we are more concerned with establishing tool chains through the use of integration technology which is tool-independent.

Thus, our main contribution will be in the definition and implementation of models and meta-models both for the targeted application domains and for existing tools. Supported by the Integration Framework, these models and meta-models will allow different tool chains to be integrated and managed. Besides, meta-modeling approaches will be used to automatically implement necessary tool chain interfaces providing the search for interoperability of tools.

Today, the solution we are developing has to face 3 issues: the interfacing between tools, the communication between tools and the orchestration of tools. For each of those issues, we have investigated several possible solutions. In the following paragraph, we present the solutions we have chosen to handle those issues. These solutions represent the basic building blocks used in the chosen architecture implementing our integration framework as illustrated in the figure 2 and detailed below.

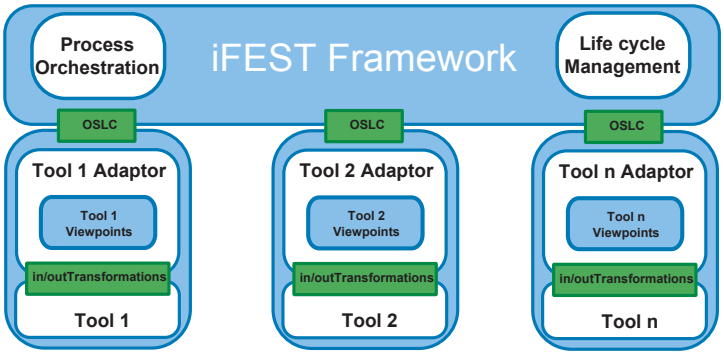


Fig. 2. Architecture of the Integration Platform

The adaptor component handles the adaptation of tools in order to facilitate its interfacing with other tools of the tools chain. The basic idea of this component is to characterize tools according to several viewpoints because we are

not only interested on how input / output data are formatted, but also on their interpretation / meaning. For instance, the figure 3 shows the main aspect of our tool viewpoint metamodel. We use then this metamodel to characterize any tool used in a co-design flow. The figure 4 shows an example of a characterization of the SpearDE tool [9] according to two viewpoints (architecture and mapping).

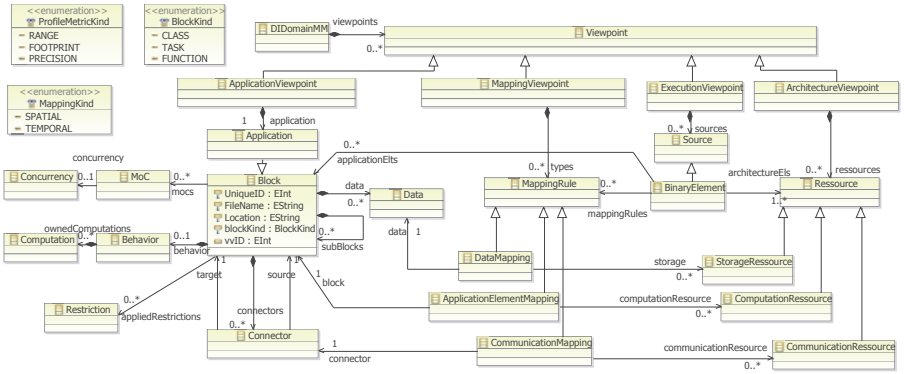


Fig. 3. Excerpt of the tool viewpoint metamodel

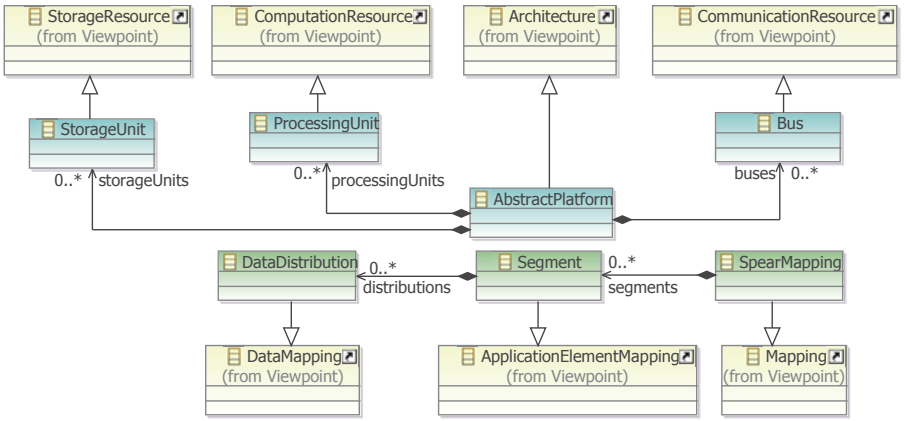


Fig. 4. Example of SpearDE tool viewpoints

While the tool viewpoint metamodel allows us to characterize the interfaces of the tools, we need mechanisms to support communication between those tools, possibly over a wide network (internet). For this purpose, we have chosen to use the OSLC (Open Services for Lifecycle Collaboration) services [1]. The OSLC

initiative is a community based effort aiming to standardize different components for tool integration. It provides a set of specifications organized into topics (Requirement management, change management, etc.), based on core specification and guidelines. The OSLC does not target tools integration, but rather specify how tool data are interfaced externally. In this approach, the resources can be provided and requested using web services.

Regarding the last point, e.g. process orchestration, we have made the choice of enacting process models. More precisely, we propose to use SPEM [12] to model co-design processes and integrate a process model execution engine into the framework in order to support a fine process orchestration, enabling or disabling tools according to the current state of the process execution. This last topic is a really hot one and is out of the scope of this paper. In the next section, we discuss what the main benefits we expect from this proposition are.

4 Expected Benefits

The proposition presented in the previous section contributes to the transition from separate sectorial, vertically structured markets to a horizontally structured market. Indeed, the open tool integration technologies enables horizontally structured market of embedded system tools as it intends to be applicable across several domains and markets. Enabling exchange of individual tools will open market opportunities for tool vendors. Enabling exchange of individual tools is also expected to stimulate cross-domain tools usage since tools used in one domain can be more easily used in the tool chain of another domain.

We expect that tools that are exchangeable in a tool chain will lower the threshold for buying tools. Thus, we think this project will also stimulate the emergence of a new innovation ecosystem around integration frameworks. Indeed, one of the final goals of this project is to produce a set of interfaces and service definitions that will be pushed to become standards. This will be facilitated by the strong links of the project partners with standards bodies. For instance, iFEST participants have been successfully involved in the standardization of MARTE and SPEM profiles.

At last, this project will demonstrate:

- a cost reduction potential of 20% for the partners' system design. It will achieve this by automating some of the processes that are not yet mainstream, such as transforms from platform independent to platform specific models, design space exploration, and broadening target hardware options (multi-core).
- a potential time to market decrease of 20% for the partners. A well-functioning tool chain will reduce both the time for setting up an effective development environment and even more the time spent for development.
- a potential of 20% for cost of poor quality for the partners. A well-functioning tool chain will avoid errors which todays manual processes introduce when going from one phase in the engineering life cycle to the next one. The tool integration solutions will enable use of analysis and synthesis tools that

increase the chances of removing errors prior to product release, and thus improving development efficiency.

The assessment of the Integration Framework will be realized according to several evaluation criteria set by the industrial partners of the project. It will be measured through the realization of real industrial use cases from different domains (software radio, aerospace, railways, etc.). Then, based on their strong experience, the industrial partners will able to say at the end of the project whether the Integration Framework has achieved its goal.

5 Conclusion

A number of interesting technologies have been studied to achieve the tool integration framework. The technologies stem from a number of different areas including model-driven engineering, and the IT and web domains. All of the studied platforms utilize variants of distributed architectures over standard protocols and middlewares. All approaches strive in different ways towards modularization and in achieving loose coupling and configurability of integration.

While some platforms have a focus on providing support for integrating engineering tools, other are more agnostic and provide basic support for integrating any type of tool. Besides those "basic" tool integration services, other services like user and project administration can be delivered as predefined services. Moreover, process modeling support is almost available in all of the platforms, at different levels of granularity. Meanwhile, process execution is still an area in progress.

In this paper, we have presented the basics of our integration framework we aim to implement and standardize. This integration framework will allow the industry to compose a tool chain by combining tools from various vendors, and even open source tools. Later, we will integrate mechanisms for process orchestration in order to finely tune the usage of the tool over the process execution. Through this work, we aim to reduce costs and time to market and improve the quality of delivered products. This work is part of the iFEST project which is a European ARTEMIS project.

References

1. Open services for lifecycle collaboration, http://open-services.net/html/Home.html
2. Aho, P., Mäki, M., Pakkala, D., Ovaska, E.: Mda-based tool chain for web services development. In: Proceedings of the 4th Workshop on Emerging Web Services Technology. pp. 11–18. WEWST '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1645406.1645409
3. Bendraou, R., Combemale, B., Cregut, X., Gervais, M.P.: Definition of an executable spem 2.0. In: Proceedings of the 14th Asia-Pacific Software Engineering Conference. pp. 390–397. APSEC '07, IEEE Computer Society, Washington, DC, USA (2007), http://dx.doi.org/10.1109/APSEC.2007.38

4. Brown, A., Feiler, P., Wallnau, K. (eds.): Past and future models of CASE integration (1992)
5. Chung, C.M., Chow, L.R., Shih, T., Wang, Y.H., Lin, W.C., Chen, J.F.: Tool integration in a knowledge abstraction environment. Information Sciences 105(1-4), 279 – 298 (1998), http://www.sciencedirect.com/science/article/B6V0C-3TKS65B-1B/2/a1f6c19d7f118884d25f1b82066dea91
6. Datta, S., Sindhgatta, R., Sengupta, B.: Evolution of developer collaboration on the jazz platform: a study of a large scale agile project. In: Proceedings of the 4th India Software Engineering Conference. pp. 21–30. ISEC '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1953355.1953359
7. Eclipse: http://www.eclipse.org
8. iFEST: Integration framework for embedded systems tools, http://www.artemis-ifest.eu/
9. Lenormand, E., Edelin, G.: An industrial perspective: A pragmatic high-end signal processing design environment at thales. In: SAMOS-III, Computer Systems: Architecture, Modeling and Simulation (September 2003)
10. Long, F., Morris, E.: An overview of pcte: A basis for a portable common tool environment. Tech. rep., CMU (1993)
11. Marquardt, W., Nagl, M.: A Model-Driven Approach for A-posteriori Tool Integration, chap. 1, pp. 3–38. Springer-Verlag, Berlin, Heidelberg (2008), http://portal.acm.org/citation.cfm?id=1422985.1422987
12. OMG: Software & systems process engineering meta-model specification, version 2. Tech. rep., OMG (2008)
13. Sriplakich, P., Blanc, X., Gervals, M.P.: Collaborative software engineering on large-scale models: requirements and experience in modelbus. In: Proceedings of the 2008 ACM symposium on Applied computing. pp. 674–681. SAC '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1363686.1363849
14. Wasserman, A.I.: Tool integration in software engineering environments. In: Proceedings of the international workshop on environments on Software engineering environments. pp. 137–149. Springer-Verlag New York, Inc., New York, NY, USA (1990), http://portal.acm.org/citation.cfm?id=111335.111346

A Domain Specific Model to Support Model Driven Development of Business Logic

Tobias Brückmann, Volker Gruhn

paluno - The Ruhr Institute for Software Technology
University of Duisburg-Essen, 45127 Essen, Germany
{tobias.brueckmann, volker.gruhn}@uni-due.de

Abstract. Despite of ongoing development of model-driven development approaches in industry and academia, we believe that there is a lack of support for the generation of business logic code from refined process models. In our work we developed a framework to support a consistent support of modeling and generation of business logic for information systems. In this article, we analyse the modeling concepts of visual behavioral modeling languages. Based on the analysis results, we introduce a domain specific model (DSM) for business logic of information systems. The DSM for business logic provides a compact and clear technical interface. It is used to connect visual modeling languages, model analysis tools, and code generators as part of our integrated framework.

1 Introduction

To provide a consistent support of model driven software development, we believe that an integrated framework considering all phases of a software process is needed (from early analysis and design until implementation and maintenance). Therefore, at least following aspects have to be considered: (1) Visual software models have to support different modeling paradigms in different levels of detail; (2) Quality assurance tasks for complex visual software models have to support modelers in preventing a faulty and inconsistent model; and (3) A clear interface between visual software models and code generators is needed, in particular if several visual modeling languages and several code generators are deployed. Despite of ongoing development of model-driven development approaches in industry and academia as described in the related work section, we believe that in particular the business logic aspects of information systems need to be better supported. In our work, we developed a framework that provides an integrated support of all phases of a development process. This includes business process modeling and system modeling, model analysis and verification of analysis and design models, generation of business logic code, and runtime assurance of the modeled business logic control flow and its constraints. The generation of structured process descriptions (such as BPEL) as required by workflow management systems are not focused (but not excluded). Additionally, we assume that the business logic of industrial information systems often depends on external functions or systems such as web services, public APIs, or further systems of an enterprise application landscape.

In our framework, a domain specific model (DSM) for business logic is used to provide a compact and clear interface between modeling languages and code generators. Based on the analysis of visual modeling languages for behavioral aspects of systems, we designed this domain specific model for business logic to support all relevant modeling concepts needed for business logic code generation of information systems. After the discussion of related work in Sect. 2, we analyse several visual modeling languages for systems behavior in Sect. 3 before we define a domain specific model for business logic in Sect. 4. Then, we give a brief overview of the usage of the business logic DSM as part of a framework in Section 5 and conclude in Section 6.

2 Related Work

In general, we follow Waddington et al. who states in [17] that „multiple modeling notations and interpretations (views) are necessary to represent each of the different aspects of concern and to fulfill different roles within MCSD [Model-Centric Software Development] such as verification of correctness, human understanding through visual interpretation, and code generation". In tool chains and frameworks for embedded systems (such as provided by Konrad et al. [8],) structural aspects are integrated with scenario models or state chart models or both, which are commonly applied behavioral modeling concepts for reactive systems. Ryndina et al. connected process and state modeling in [13] through generation of object live cycles based on process models, but they do not focus on an abstract model providing an integrated view. Gerth et al. provide in [4] a so-called "intermediate representation (IR)" of process models as abstract representation of process models and applied it for model change management purposes. However, this approach do not consider structural or state modeling concepts. Mohan et al. provides with their FlexFlow model in [9] an integration of structural and state modeling, but they do not consider process modeling aspects. However, for business logic generation for information systems we need integrated model supporting process modeling (containing all actions and control flow specification), structural modeling (containing all business objects that are relevant during process execution) and state modeling (containing domain states of business objects and object live cycles).

In context of modeling systems behavior there are several approaches that transform visual behavior diagrams into special purpose languages. One purpose for such transformation is to get a formal-founded structure for automated model quality assurance. Target models are for example process algebras (e.g. Engels et al. in [3]) or Petri nets variants (as discussed by Stains in [15]) which are suitable for automated reasoning or simulation. However, the high level of abstraction of formal models makes them difficult to handle them as technical interfaces for business logic code generators. Otherwise, in case of using domain specific behavioral models for code generation, they are designed for proprietary tool chains (such as FlexFlow [9] or J3 [18]) or specific target platforms (such as BPEL in [7], or Embedded Systems e.g. in [8]). As a difference to related approaches,

with our DSM for business logic, we aim to support a compact and well-defined interface to connect different modeling languages, model transformation rules, and code generators without any dependency to modeling languages or target platforms. Approaches such as [2] also define and use intermediate models for behavior modling of information systems. However, we explicit consider state modeing in form domain states in our approach.

3 Analysis of Visual Behavior Models

The following analysis of visual behavior models provides the basis for our domain specific model for behavior logic. For the analysis of major behavioural modeling concepts, we considered following visual modeling languages for process and systems behavior: eEPC [14], BPMN [11], Petri Nets [12] and Coloured Petri Nets [6], YAWL [16], and UML [10]. We dropped MSC [5], pure State Charts, and Data Flow diagrams for detailed analysis: MSCs focus only on scenarios and pure State Charts focus only on states. Data Flow diagrams contains a graphical depiction of data flow through a system, but only from an abstract point of view. Model details such as order of processes, or control flows aspects are missing.

Fig. 1 provides an tabular overview of modeling aspects and their support by visual modeling languages. In our analysis we focus on aspects of process modeling, structural modeling and state modeling concepts and their integration.

Process Modeling Each analysed modeling language supports an atomic element that contains functions/actions/tasks that are executed during runtime of a process (see the first row in Fig. 1). It is named "function" in EPC, "task" in BPMN and YAWL, "action" in UML, and "transition" in Petri Nets. Moreover, "nesting" provides an opportunity for a more usable visual and logical structure that also allows the reuse of parts of the model. In eEPC such elements are called "process connector", in BPMN "sub-process", in YAWL "composite task", and in UML "activity". In Petri Nets, net transformations such as "coarsening" and "refinement" can be defined to support nesting.
Flow control elements are used to specify in which order and under which conditions executable items as well as nested parts of the model are processed at runtime. Flow control elements have usually either one incoming and several outgoing connections ("split") or one several incoming and one outgoing connection ("merge"). Typically, XOR, AND, and OR nodes are used to merge or split different control flows. Such flow control nodes are supported by high-level modeling languages eEPC, BPMN, YAWL, and UML. Petri nets (as well as CPN) do not support flow control nodes. A further concept to specify flow control are constraints that are associated to control flow edges. Coloured Petri Nets use so-called "arc functions" as an important flow control concept. The UML also use "guards" as constraints in activity diagrams. EPC, BPMN and YAWL do not include "guards" of control flow edges. However, YAWL introduces "conditions", which can be compared to places in Petri nets and which are used to specify

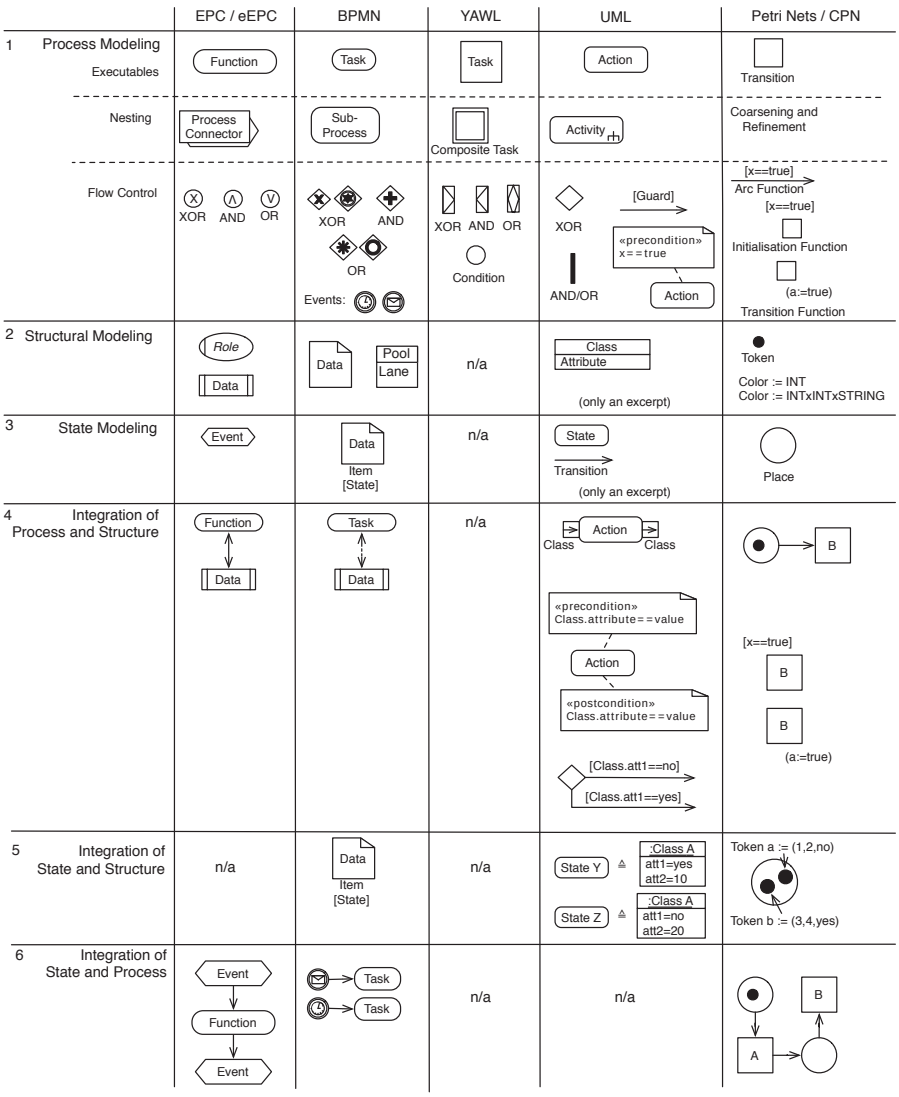


Fig. 1. Process Modeling Concepts supported by Visual Modeling Languages

patterns such as mutual exclusion. BPMN supports different types of “events” that are used to model conditions for task or sub-process execution. Constraints are also used to specify conditions that have to be assured during process execution. In CPN constraints are named “initialisation functions” and in UML they are defined with “preconditions” and “postconditions”.

Structural Modeling As shown in Fig. 1 eEPC and BPMN supports structural modeling only on a very high level of abstraction: In eEPC “roles” and “data” items can be specified, whereas a “role” determines a user role or an organizational unit that is responsible for a function and a “data” item describe information that is used or provided by a function. BPMN supports the “data” concept comparable to eEPC: A “pool” is used to identify an organization and a “lane” is comparable to “role” of eEPC. YAWL, which is defined as a workflow modeling language, does not support structural modeling concepts. CPN allows to specify different types of tokens (called ”colours”). Each token can comprise different properties (including composite properties), so that the set of current attribute values describes the internal state of tokens in CPN. The most complex structural modeling concepts are supported by UML class diagrams. For illustration purposes, Fig. 1 contains only a very basic set of structural model elements as supported by UML: classes (used to specify entities) and attributes (used as properties of classes).

State Modeling In general, states can be used in different scopes: a single state can indicate the actual internal state of system as a whole (system scope), or the internal state of an object (object scope), or the internal state of a property of an entity (property scope). State modeling in eEPC is applied on system scope, each function leads to an “event”, which is used to defined the current state of a process. The state of a Petri net is described by its marking (defined by number of tokens in all places). With the use of CPN, single tokens can be distinguished by their types and attribute values, so that state modeling at object level is also supported by CPN. UML provides a separate view for state modeling (state machine diagram) at system scope and object scope. It comes with a set of convenience elements (such as history states) that reduces the visual complexity in large state diagrams. States in BPMN can be defined only together with data items. YAWL do no support state modeling.

Integration of Process and Structure In eEPC and BPMN structural entities (“data” items) can be linked to functions or tasks. The link is established by a directed edge, which also enables a bi-directional connection. The connection between functions and data items is used to specify that a data item is used or modified or both. In UML, such a dependency is expressed with input and output pins that are modeled in relation to an action (or with an explicit modeled data flow). In contrast to eEPC and BPMN, UML data items (classes and attributes) can be directly linked to flow control elements (such as guards or preconditions) as well as to local constraints (such as postconditions). The integration of structural and process modeling in CPN is done by initialization and

transition functions, which can be specified for each transition. Comparable to UML guards arc functions in CPN are also be linked to tokens directly. YAWL does not provide any integration between structural and process modeling concepts.

Integration of State and Structure Integration between state and structural aspects of a model is only considered by UML and Coloured Petri Nets: In UML states of a state diagram can be linked to an internal state of either an object (object scope) or an internal state of a whole system (system scope). In CPN places are typed, they only contain tokens of a defined type. Hence, the marking of a CPN depends on the type of tokens, which is related to internal state of a token. Furthermore, arc functions can be used to specify dependencies between the internal state of a token and its routing to a place. States in BPMN are always part of defined data items. EPC and YAWL do not support integration of structural and state modeling.

Integration of State and Process State modeling and process modeling concepts in EPC are realized through the alternating sequence of events and functions: a function follows an event (which is actual a system state) and results in an event (a further system state). Hence, states and processes are directly linked. The same concept of alternation between executables and system states is realized by Petri Nets, where transitions and functions alternate. A further way to link states and processes is used in BPMN, where events trigger the processing of executables. UML and YAWL do not provide explicit integration elements for states and process.

4 Abstract Business Logic Model Definition

Based on the analysis result of visual modeling languages in the previous section, this section provides the definition of our abstract business logic model (Amabulo model). A general overview of all supported meta model elements is given by the class diagram in Fig. 2.

The dashed lines group elements of three supported modeling concepts: Elements for **process modeling** are needed to describe the business logic in form of structured sequences (nesting) of executables and their dependencies. They include control flow elements, required parameters, and conditions. In an Amabulo model "user functions" are elements that specify atomic process elements (executables) that are executed by users. "System functions" are atomic process elements (executables) that are executed automatically by a system. As modeled in Fig. 2 both types of executables can be generalized to "function". "Processes" are used for nesting of functions. Processes and functions can be generalized to so-called "activity concepts". An activity concept can have specified input and output "parameters". Moreover, "constraints" can be used to define preconditions and postconditions for an activity concept. "Succession" elements are used to determine the sequence of activity concepts depending on control flow constraints.

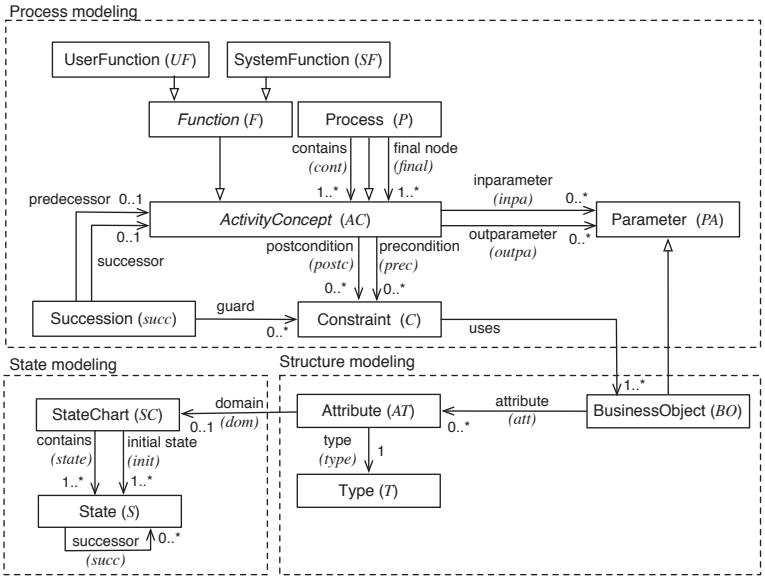


Fig. 2. Amabulo Meta Model

Elements for **structural modeling** are needed to define business objects and their properties as used by control flow constraints to determine the actual flow through a process. Moreover, objects are used by functions to get information about the current business data set, or to modify business data, or both. As Fig. 2 shows, "business objects" in an Amabulo model are specialized parameters. Business objects can have "attributes" and each attribute has a defined "type".

Elements for **state modeling** are used to refine attributes of business objects by specification of object live cycles with the use of domain states. Therefore "state charts" are elements of an Amabulo model that use "states" for domain state modeling (at property scope).

More precisely, an **Amabulo model** A is denoted as tuple $A = (M_A, R_A)$ and consists of two different sets: M_A the set of sets of different modeling concepts, which contains a set of user functions UF, a set of system functions SF, a set of all functions F with $F = UF \cup SF$, a set of processes P, a set of all activity concepts AC with $AC = F \cup P$, a set of parameters PA , a set of constraints C, a set of business objects BO, a set of attributes AT, a set of types T, a set of state charts SC, and a set of states S.

The symbol R_A of an Amabulo model indicates a set of relations between elements of elements of M_A (such as relations between attributes and business objects or functions and processes). As shown in Fig. 2 those relations are namely a relation *type* that specifies the data type of an attribute, a relation *value* that defines the actual value of an attribute, a relation *att* that relates attributes and business objects, a relation *dom* that relates attributes and state charts, a

relation $succ_S$ that defines state transitions, a relation *init* that defines initial states of a state chart, a relation *cont* that relates processes to its contained activity concepts, a relation $final$ that defines final activity concepts of a process, a relation $succ_{AC}$ that defines an order between activity concepts, a relation *prec* defining preconditions of activity concepts, a relation *postc* defining post-conditions of activity concepts, a relation *inpa* that relates input parameters and activity concepts, and a relation *outpa* that relates output parameters to activity concepts.

5 Using The Abstract Business Logic Model

The previously developed abstract business logic model is used as an important interface of our architectural blueprint (which we call "Amabulo infrastructure"). An Amabulo infrastructure comprise models, meta models, model transformations, and tools to set up the technical project infrastructure for model driven development of business logic of information systems. As shown in Fig. 3, the framework consists of five different layers: The **Visual Model Layer** is re-

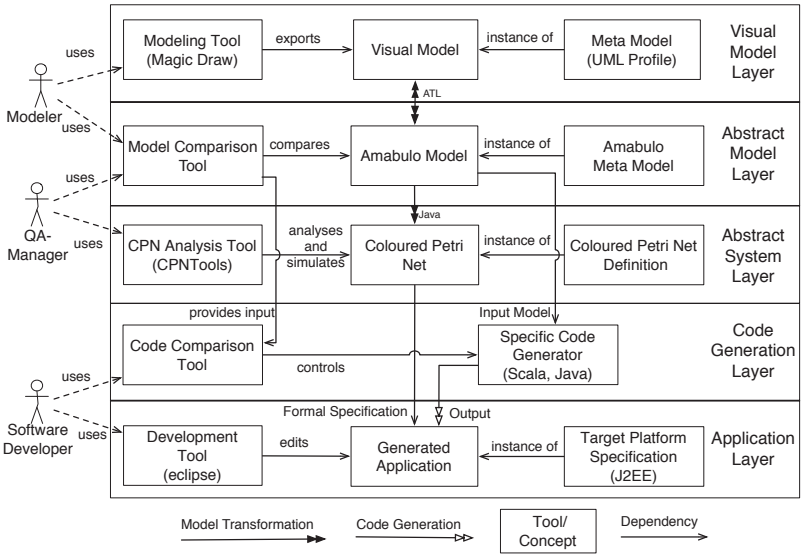


Fig. 3. Overview of the "Amabulo Infrastructure" Framework

sponsible for all visual modeling tasks. The visual modeling language and the modeling tool of this layer are used by human modelers to specify domain and technical requirements. Depending on actual requirements, yet existing modeling languages (such as UML [10], BPMN [11], or EPC [14]) and tools can be reused. The Visual Model Layer supports multiple modeling language in one concrete

project setup: In early phases when talking to domain experts, a business process modeling language (such as EPC) can be the best choice for visual modeling. Afterwards, when domain requirements are complete and a technical modeling language is needed, the Visual Model Layer can be switched to Unified Modeling Language (UML). After the manual modeling tasks, the visual model is automatically transformed into an Amabulo model, which is an instance of our business logic DSM. The **Abstract Model Layer** provides an abstract (non-visual) view onto the business logic model as introduced above in Sect. 4, which is reduced only to model elements that are revelant for business logic generation. It is used as a common interface between visual modeling languages and code generators. With only having 13 precisely defined model elements it helps to reduce the complexity of the interface between models and code generators. Moreover, the abstract model is used to assist users in comparing different versions of the same model and exploring semantic changes between them as needed for impact analysis during maintenance tasks. Additionally, we implemented an automated model transformation from the abstract model into Coloured Petri Nets as part of the **Abstract System Layer.** Such a generated Coloured Petri Net representation of the modeled business logic is a formal structure that can be analysed and simulated by quality assurance managers with suitable analysis and simulation tools (such as CPNTools). The **Code Generation Layer** as introduced in [1] is responsible for generation the business logic of information systems at the **Application Layer**.

We implemented a proof of concept using an UML profile as visual model and JBoss-Seam as target platform for code generation. After the first experiences with this proof of concept infrastructure, we need to test our approach in scenarios with different modeling languages and different target platforms for the business logic. Therefore, we currently integrate BPMN as a further modeling language and a .NET code generator.

6 Conclusion

In this article, we analysed visual modeling languages for business processes and systems behavior. Based on this analysis, we developed a domain specific model for business logic. This DSM provides a compact and clear technical interface to connect visual modeling languages, model analysis tools, and generators for business logic for information systems. The developed business logic model is used as a central artifact in a framework, which was developed to support a consistent model driven development process of business logic for information systems.

References

1. Tobias Brückmann and Volker Gruhn. An architectural blueprint for model driven development and maintenance of business logic for information systems. In *Proceedings of the 4th European Conference on Software Architecture (ECSA2010)*, 2010.

2. Raymonde Le Delliou, Nicolas Ploquin, Mariano Belaunde, Reda Bendraou, and Louis Féraud. A model-driven approach for information system migration. In *Proceedings of the 4th Workshop on ODP for Enterprise Computing*, 2004.
3. Gregor Engels, Jochen Küster, Reiko Heckel, and Luuk Groenewegen. A methodology for specifying and analyzing consistency of object-oriented behavioral models. In *ESEC/FSE-9 Proceedings of the 8th European software engineering conference*. ACM, 2001.
4. Christian Gerth, Jochen Küster, and Gregor Engels. Language-independent change management of process models. *Model Driven Engineering Languages and Systems*, pages 152–166, 2009.
5. International Telecommunication Union (ITU). Message sequence chart (msc). http://www.itu.int/rec/T-REC-Z.120/en [last checked 2010-06-15], 2004.
6. Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*. Springer-Verlag, 1992.
7. Jana Koehler, Rainer Hauser, Jochen Küster, Ksenia Ryndina, Jussi Vanhatalo, and Michael Wahler. The role of visual modeling and model transformations in business-driven development. In *Proceedings of the Fifth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2006)*, volume 211 of *Electronic Notes in Theoretical Computer Science*, pages 5–15. Elsevier Science, 2008.
8. Sascha Konrad, Heather Goldsby, and Betty Cheng. i2MAP: An Incremental and Iterative Modeling and Analysis Process. In *Model Driven Engineering Languages and Systems (MoDELS 2007)*, volume 4735/2007 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.
9. Rakesh Mohan, Mitchell Cohen, and Josef Schiefer. A State Machine Based Approach for a Process Driven Development of Web-Applications. In *Advanced Information Systems Engineering (CAiSE 2002)*, volume 2348/2006 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
10. Object Management Group (OMG). Unified Modeling Language (UML): Superstructure, Version 2.3. http://www.omg.org/spec/UML/2.3 [last checked 2011-04-15], 2010.
11. Object Management Group (OMG). Business Process Modeling Notation (BPMN) 2.0. http://www.omg.org/spec/BPMN/2.0 [last checked 2011-04-14], 2011.
12. Carl Adam Petri. *Kommunikation mit Automaten*. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.
13. K Ryndina, J Küster, and H Gall. Consistency of business process models and object life cycles. In *MoDELS 2006 Workshops*, 2006.
14. August-Wilhelm Scheer, Oliver Thomas, and Otmar Adam. Process modeling using event-driven process chains. In *Process-aware information systems: bridging people and software through process technology*. John Wiley and Sons, Inc., 2005.
15. Tony Staines. Intuitive mapping of uml 2 activity diagrams into fundamental modeling concept petri net diagrams and colored petri nets. *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2008)*, 2008.
16. Wil van der Aalst and Arthur ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245 – 275, 2005.
17. Daniel Waddington and Patrick Lardieri. Model-centric software development. *IEEE Computer*, 39(2):28–29, Feb 2006.
18. Jules White, Douglas Schmidt, and Aniruddha Gokhale. Simplifying autonomic enterprise java bean applications via model-driven engineering and simulation. *Software and Systems Modeling*, 7(1):3–23, 02 2008.

Towards an Enactment Mechanism for MODAL Process Models

Pierre-Yves Pillain, Joel Champeau, Hanh Nhi Tran

Ensta-Bretagne
2 rue Francois Verny, 29806 Brest cedex, FRANCE
{pierre-yves.pillain, joel.champeau, hanh_nhi.tran}@ensta-bretagne.fr

Abstract. Process modeling is mainly oriented towards static descriptions of processes which are operated through another framework and/or language. In these cases, the process execution is generally loosely controlled and it is hard to verify the satisfaction of process' intention. We propose an enactment mechanism for the process modeling language MODAL that allows a smooth transition from process modeling to process execution and enable a tight integration of development tools as well as a validation of process results.

Keywords: process execution, process modeling, process monitoring.

1 Introduction

Awareness of process and its improvement has increased significantly over the past decades in information systems and software development. Both communities try to develop methods, techniques, and tools to support the design, enactment, management, and analysis of operational processes. Where the application of technology to support process has reached a certain maturity level in the Business Process Management domain, it is still in its early days for Software and System Process. However, some efficient mechanisms to assist, to control, and to monitor complex processes in the software/system development are certainly useful.

Business Process Management is a well established research and practice field which aim to provide the assistance for operational business processes from the design to the enactment phases. Thus, works in this field have been investigated on both business process modeling and business process execution as well as on the transition from modeling to enacting. The consolidation of the community has led to a single language for business process execution BPEL [9] and the well adopted standard business process modeling notation BPMN [12]. These results enable the development of several commercial workflow and business process management systems that can support a complete business process lifecycle.

Software Process Engineering, in contrast, generally separates process modeling from process enactment and does not always define the connection between these two axes. Most of works in this domain focus on developing software and system process modeling languages but few define mechanism to execute process models [2][4]. Consequently, software process models are generally not executable and software process automation tools have been remarkably absent.

To remedy the deficiency of software process enactment mechanisms, there were attempts to benefit from the execution features implemented in business process engineering tools to enact software process models (for instance, the works in [1][2] or the suggestion in [9]). However, for dealing with software and system processes, which are usually very complex and unstable, the execution mechanism provided by business management process systems seems inadequate. Generally software processes concern with several iterations, several versions of products and use many development tools. Workflow systems support well the tasks sequencing in processes, but they are not equipped with necessary constructs to allow a tight controlling of software development activities, e.g. for managing and validating software artifacts or for integrating development tools. Therefore, we believe that proper mechanisms for software process enactment are necessary and should be more investigated.

Our objective is to develop a process centered environment that support Model Based Engineering (MBE) processes. Such an environment should provide tightly interwoven modeling and enacting modules in order to enable a coherent management of process definition and execution. In the previous work [7], we proposed the process modeling language MODAL, a SPEM2.0 [9] extension dedicated to specify MBE processes. In this work, we propose the extensions to MODAL and implement the necessary mechanisms to enact MODAL process models. To enable MODAL process' execution, first we added to MODAL the concepts allowing modeling the behaviors of activities and process components. We also modified the Java Cometa Framework [7], an extension of MARTE [11] that abstracts hardware platforms to communication models, to take into account the support for process enactment. Then, we defined a transformation converting a MODAL process model to a Cometa program in order to generate the process's executable code. This transformation is integrated and realized in a process engine prototype which allows loading MODAL process models, instantiating them into process instances and then executing different process instances.

In the next section we present the main concepts of MODAL process modeling language and the extensions made to describe process execution models. In Section 3 we explain our approach to support the execution of MODAL process models. Section 4 presents the process engine prototype being developed together with a case study illustrating and validating our propositions. In the conclusion, we give some brief discussions on related works, and on our ongoing works

2 MODAL Process Modeling Language

MODAL (Model Oriented Development Application Language) [14] is a SPEM 2.0 based process modeling language which introduces additional concepts enabling "Model Based Engineering" (MBE) process definition and elicitation and reinforces the semantics of process models to enable some degree of process executability. In this section, first we outline the philosophy and contributions of MODAL, then we present the main MODAL concepts needed to describe executable process models. The detail description of MODAL can be found in [14]. Fig. 1 shows a fragment of the metamodel representing the main MODAL concepts.

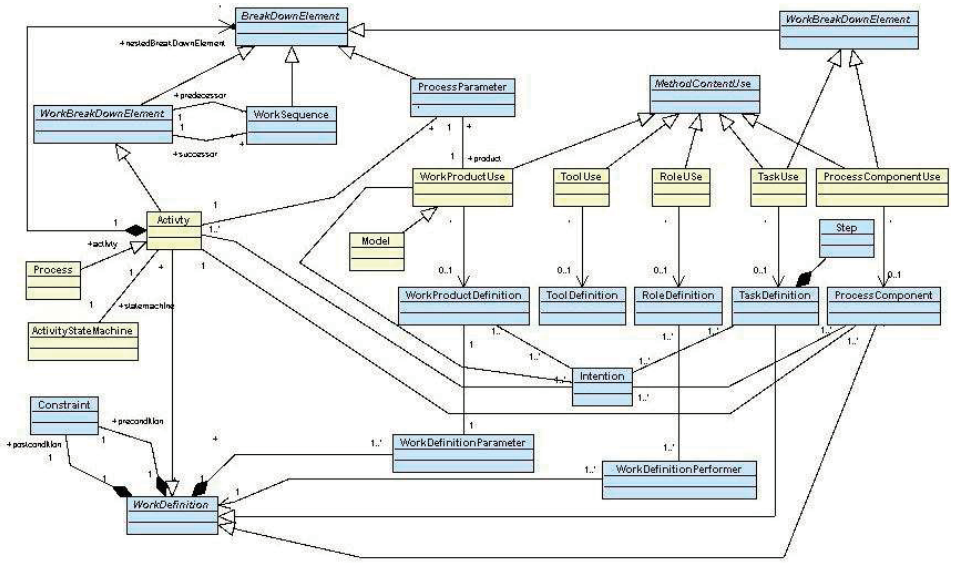


Fig. 1. MODAL core concepts

As a SPEM 2.0 extension, MODAL separates reusable method contents from their applications in processes. In the method content space, MODAL keeps the basic elements *TaskDefinition*, *WorkProductDefinition*, *RoleDefinition* from SPEM 2.0 and defines, clarifies the concepts of *ToolDefinition* and *ProcessComponent*. In software process, especially in MBE process, tools play an important role in the development and should be integrated into process description and execution. The concept *ToolDefinition* is thus introduced to facilitate the specification of tools and ther integration to development process. One important contribution of MODAL is the clarification of *ProcessComponent* concept. When *TaskDefinition* of SPEM allows the reuse of "white-box" and non executable activities, MODAL provides a precise definition for *ProcessComponent* concept to allow the capture of "black-box" executable activities. Therefore, *ProcessComponent* enables a more flexible way to define and reuse process as well as to integrate tightly development tools to process' execution. The relations between *TaskDefinition* and *RoleDefinition*, *WorkProductDefinition* are established via the relations *WorkDefinitionPerformer* and *WorkDefinitionParameter*. *ProcessComponent* defines the relations with other elements through its ports.

Like SPEM 2.0, in the process space, MODAL provides the concepts representing process elements defined in the process or reused from method content elements. The concepts in MODAL process space are *TaskUse*, *WorkProductUse*, *RoleUse*, *ToolUse* and *ProcessComponentUse*. A MODAL process is composed of activities. An *Activity* (from SPEM 2.0) represents a grouping of nested breakdown elements such as *RoleUse*, *WorkProductUse*, *TaskUse*, *ProcessComponentUse* or other activities. An activity defined directly in a process can express its relations with *WorkProductUses* and *RoleUses* by using *ProcessParameter* and *ProcessPerformer*. In cases of reuse predefined tasks or components (via *TaskUse* or *ProcessComponentUse*), the relations between process elements are inherited from their corresponding definition elements in the method content space (i.e. from *TaskDefinition* or *ProcessComponent*).

To support MBE process modeling, MODAL aligns the definition of work product to model to allow specifying the metamodel typing a model as well as the model's lifecycle and inter-model relationships. MODAL also formalizes the *Constraint* concept associated to process activities to insure the consistency of the process. Thanks to the refinement of work product definition, the pre and post-conditions of an activity can be better expressed as the constraints defined on products (models) manipulated by the activity. The execution of process activities hence can be guarded by more formal constraints insuring a finer grained causality in the definition of processes. This approach is of great importance in the context of MBE processes where wide scope languages are used.

One of MODAL's special features is the separation of process' intentions from their realization to facilitate process definition and analysis. MODAL introduces the concept *Intention* defining a methodological objective set by various stakeholders. When defining an intention, MODAL allows specifying and putting together several aspects of process need on products (business rules), tools (technical constraints) and required standards (organization constraints) for example. An intention can be satisfied by one or more strategies representing different technical solutions. MODAL defines the relationship "satisfy" between the intentions and creates intention maps to guide process designers choosing an appropriate strategy in a specific technical space to realize a process satisfying all intentions.

An intention can be associated to a work product definition representing the result obtained when the intention is satisfied. This relation can help classifying work products according to their intention, but it cannot allow verifying if a concrete work product obtained during the process' execution satisfies the associated intention. To enable such a validation, we associate an intention to an activity (or a *TaksDefinition*, a *ProcessComponent*) and assure that the activity's output product and the intention's product is the same. When executed, the activity's post-conditions expressed on the output product's content will be automatically verified to validate the satisfaction of process intention associated to the activity.

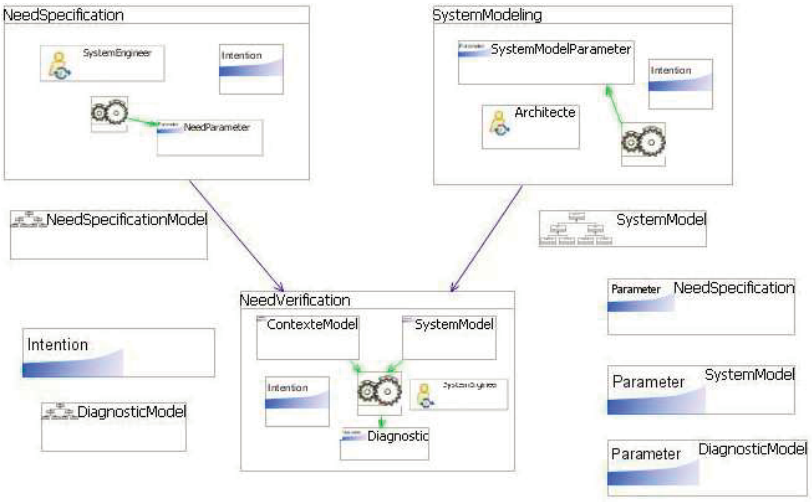


Fig. 2. Trustworthy specification process

Fig. 2 shows an example of a MODAL process model composed of three activities: *"NeedSpecification"*, *"SystemModeling"* and *"NeedVerification"*. These activities are linked by *WorkSequences* of type *"finishTostart"* so the *"NeedVerification"* activity just can start if both the *"SystemModeling"* and *"NeedSpecification"* activities are terminated. Each activity satisfies an intention and contains several process elements like roles, workproducts and process components. For example, the *"NeedSpecification"* activity contains an output named *"NeedParameter"* referencing *"NeedSpecificationModel"* model and the *"NeedSpecificationEditor"* process component. This activity is realized by the role *System Engineer*.

3 Support MODAL Process Enactment

In order to enable MODAL process' execution, we realize a transformation converting a MODAL process model to an executable process model. We choose Cometa Framework [6][7] to simulate process model instances. Cometa is a language describing models of computations (MoCs) and helping to design a virtual execution platform for the simulation of hierarchical concurrent components communicating; therefore Cometa allows us to define the process's execution semantics independently of the semantics of a specific platform (i.e. process engine). Cometa has been implemented as a library of java classes representing its different concepts. The execution platform is described as a class that instantiate all the elements of the virtual platform and how they communicate. Fig. 3 shows the general schema for transforming a MODAL process to an executable process.

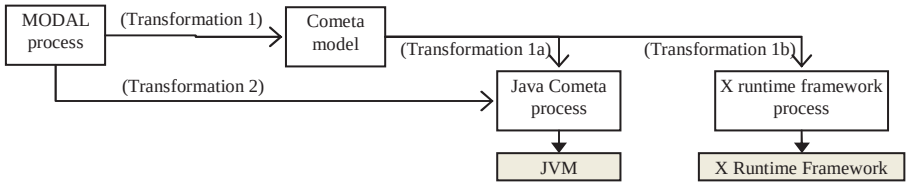


Fig. 3. Transformation from MODAL process model to executable process model

First, we transform MODAL process model into a Cometa model defining process' execution semantics on a virtual platform (Transformation 1). In principle, from this Cometa process model, we can generate different executable process codes for different process execution platforms, e.g. for the Java Virtual Machine (Transformation 1a) or for a X runtime framework (Transformation 1b). Because both the Cometa framework and process components are implemented in Jave, in our prototype's implementation, we take a "shortcut" transformation to generate directly an executable Java program from a MODAL process (Transformation 2). To support such a transformation, first we extend MODAL with the concepts modeling process' behaviors. We also added to Cometa the necessary constructs related to execution modeling to enable the adequate transformation from MODAL processes to Cometa programs.

3.1 MODAL Process Execution Model

To enact a process model, we have to know its execution semantics. Thus, we extended the MODAL metamodel proposed in [7] to allow the description of process execution models. In MODAL, the behavior of a process is described at three levels: the workflow

of process' activities; the behavior of an activity; and the executable actions of a process component. The process workflow is already described by the SPEM relation *WorkSequence* between activities. The behavior of an activity defines how its instances change their states during the process execution.

We use a state machine to model the state transition of an activity's instance (Fig. 4). Besides the principal activity's states *"Start"*, *"Run"*, *"Finish"*, *"Abort"*, we proposed two other states *"Repeat"* and *"NotValid"* to represent situations where an activity is repeated because it can or cannot satisfy its intention. As explained in the section 2, we incorporated the validation of activity's intention – expressed through the constraints on its work products- in order to better control the process' execution. When changing to a "*Run*" state, an activity is realized by the execution of the *ProcessComponent* referenced in the activity's *ProcessComponentUse*. An activity can reach "Finish" state when its performer can validate activity's work products, in the context of MBE, such a validation concerning a verification of a set OCL rules on output models.

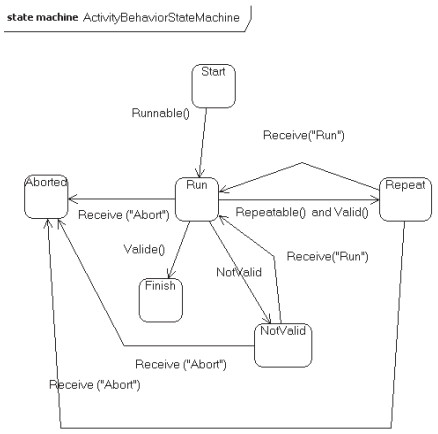


Fig. 4. Activity's state machine

To make process components executable, we define an action language allowing the description of a process component's internal behavior (Fig. 5). The component's behavior is modeled by a flow of executable Actions. These actions can perform communication instructions (*SendAction*, *ReceiveAction*) or invoke an executable component (*Execute*).

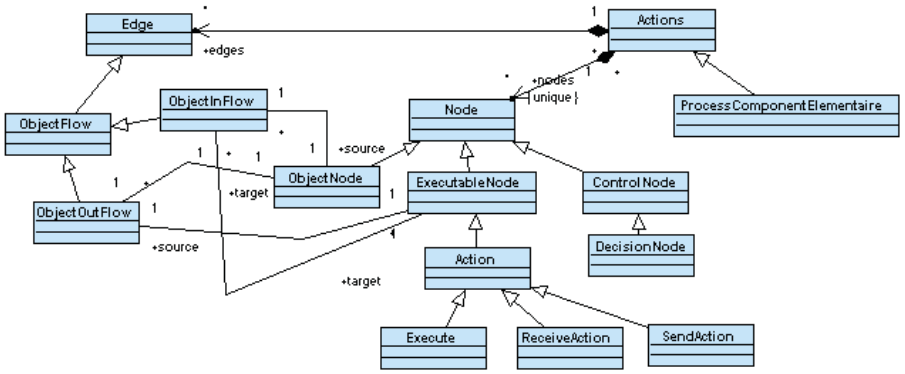


Fig. 5. Definition of action language associated to a process component

3.2 Cometa Framework

In Cometa, a MoC domain gathers the definition of a particular model of computation regarding the orthogonal concerns on the behavioral scheme, the temporal scheme, the data scheme, and the communication scheme. With Cometa, the behavior during

communication phases between two system blocks is represented in the components *MoCComponent*, which is a concurrent entity having ports (*MoCPort*) and connectors (*MoCConnector*), in which application blocks are allocated. The behavior of the underlying model of computation is given by a *Domain* and the elements that describe the *Behavior* of a specific MoC.

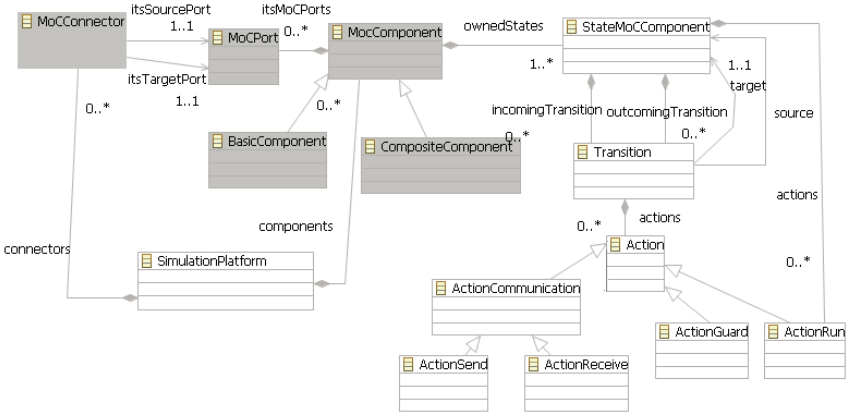


Fig. 6. Main Cometa concepts

We want to simulate each MODAL activity by a Cometa *MoCComponent* thus simulate the process execution by activities communications. In MODAL, we modeled the behavior of an activity by a state machine. So, we added to Cometa the necessary constructs to implement such a state machine. The Fig. 6 shows a fragment of Cometa metamodel defining the main concepts used to enable process execution (the grayed elements represent the original Cometa concepts). We use the concept *SimulationPlatform* to describe the process' topology: how its activities are interrelated. The *MocComponent* represents an activity, its behavior can be described thanks to the other concepts: *StatMoCComponent*, *Transition* and *Action* which implement the activity's state machine. Especially, the *ActionRun* is added to enable the execution of executable components captured in the activity's process components.

3.3 Transformation MODAL-Cometa

Having the MODAL process metamodel and the Cometa framework which enable the description of process execution model, we can define the transformation from a MODAL process model to a Cometa executable program. The transformation covers the following process's aspects: structure, behavior description and execution semantics.

Structure: From a given MODAL process model, we define these following transformation rules to generate its corresponding COMETA program: (1) *Process* is transformed into *SimulationPlatform; (2) Activity* is transformed into *BasicComponent; (3) WorkSequence* is transformed into *MoCConnector* and *MoCPort*; (4) *Execute* action of *ProcessComponent* is transformed into *ActionRun.* The transformation creates one *BasicComponent* for each *Activity* and then creates the process' topology.

Behavior and Execution Semantics: As presented in section 3.1, the behavior of an Activity is described by a state machine. When an activity is transformed into a Cometa *BasicComponent*, we add to the generated component a state machine (c.f. Fig. 4).

Current *state* attribute describes current state reached by an *Activity*. Activity behavior state machine describes reachable states: *"Start"*, *"Finish"*, *"Abort"*, *"NotValid"* and *"Repeat"*. Communication between *Activity* are necessary to perform control event and data. We define therefore the *"Finish"*, *"Repeat"*, *"Valid"*, *"NotValid"* control message. An activity can send these messages to all of its successor activities by a KPN (Kahn Process Network) transmission. When a process model instance starts, all activities are in the state *"Start"*. An activity can step into *"Run"* state if the *"Runnable"* guard is set, i.e. if the activity has no predecessor activity or it receive a message *"Finish"* or *"NotValid"* from its predecessors (supposed that all the *WorkSequences* are *finishTostart* type). *"Valid"* guard enables an activity to change to *"Finish"* or *"Repeat"* states. This guard's value is set to *True* when the OCL constraints associated with the simulated activity is validated. By allowing to represent these constraints as the conditions on products manipulated by the executable activity's *ProcessComponent* and to evaluate them automatically, we can control if the activity is correctly performed to satisfy its intention.

4 MODAL Process Engine Prototype

We are developing a process environment including several tools to support process modeling and simulation. These tools are developed in Java with EMF, GMF and SWT APIs. We outline in this section the functionalities of the MODAL Process Engine Prototype and illustrate the use of this environment with a case study.

4.1 MODAL Process Engine

The Fig. 7 presents the different steps of process simulation in the MODAL Process Engine that is being developed: first we use the *MODAL editor* to create process model; then we instantiate a process model to obtain one or more executable process instances by using *Process Instantiation Editor*; finally the process instances are executed on the *Java Execution Engine*.

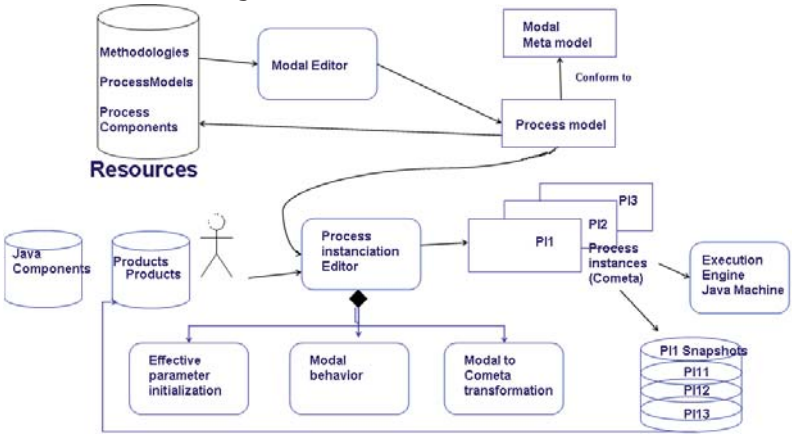


Fig. 7. MODAL Process Engine

MODAL Editor: This tool helps process designers to create MODAL process models. Process designers can use reusable process elements (process models, process components) in the Methodologies base when modeling a process.

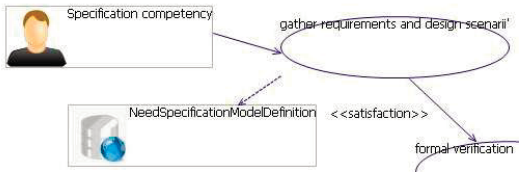
Process Instantiation Editor: This tool allows creating executable process instances from a process model by substituting process formal parameters with effective parameters, i.e. the physical resources generating process instances' Cometa codes. To do so, we provided three sub-modules to (1) refer to physical resources (concrete models and real actors) for formal process parameters concerning workproducts and roles; (2) complete the MODAL model with the process behavior model; (3) transform the MODAL process model into Cometa programs. Transformation rules are implemented in Java.
Simulate Process Instances: Cometa process instances are simulated by the Java Virtual Machine. During the process enactment, we allow to serialize process instances as process snapshots to keep trace of process execution.

4.2 Case Study

The case study illustrates the use of MODAL Process Engine to design a process whose intention is to produce a verifiable specification of needs. Thus, this specification must be consistent in order to be checked by a formal tool. The designed process is generic and does not depend on a particular domain. We do not present here the whole process modeling phase, just give in Fig. 2 the MODAL process model created with our MODAL editor. Below, we focus on the definition, instantiation and simulation of the "*NeedSpecification*" activity.

The Fig. 8 shows that the *WorkProductDefinition* named *Need SpecificationModelDefinition* is linked to the intention "*gather requirements and design scenarii''*. To achieve this intention, we defined the activity *NeedSpecification* which produces a model conformed to the *NeedSpecificationModel*.

Fig. 8. *NeedSpecification* Intention



In order to define an activity, in MODAL a method engineer has to choose a technological space specifying the workproducts manipulated by the activity and the process component which performs the activity. In our case study, we use the *Process Instantiation Editor* to realize this step (Fig. 9).

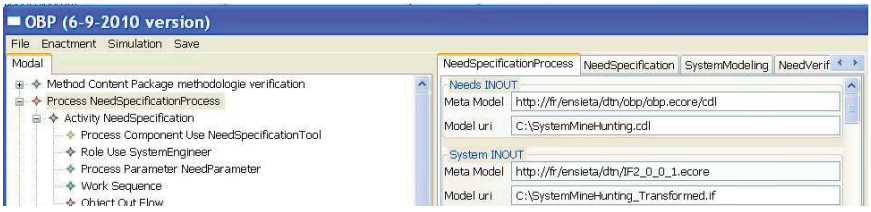


Fig. 9. MODAL Process Instantiation Editor

The *NeedSpecification* activity has one parameter of type "inout" which is defined by the *NeedSpecificationModelDefinition* and refers to the *NeedSpecificationModel*.

At this step of process instantiation, we have to specify the language used to represent the activity's workproduct. In this example, we chose the metamodel CDL (Context Description Language) [13] to represent the *NeedSpecificationModel*. The result of this *NeedSpecification* activity is thus a model conformed to CDL metamodel. This choice of this metamodel determines the choice of the *ProcessComponent* performing the activity. So we chose the *NeedSpecificationEditor* process component which allows to produce a CDL model specifying system's needs. As explained, the model created by the *NeedSpecification* activity must satisfy the intention "*gather requirements and design scenarii*". To enable a validation of intention satisfaction, an OCL constraint is defined to verify the content and states of the output model. For example, here we defined the following constraint on *NeedSpecificationModel*:

```
self.topActivity->size()>0  //the CDL model contains at least one actor.
```

The next step is to create an executable instance of *NeedSpecification* activity by realizing the MODAL-Cometa transformation. The result of this transformation (Fig. 10) shows that each *BasicComponent* contains the state machine incorporated during the transformation.

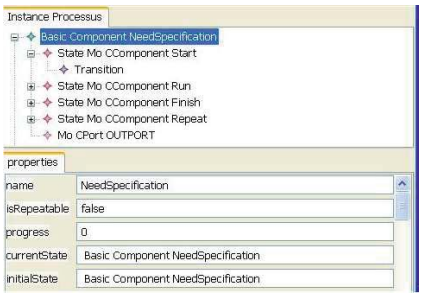


Fig. 10. An instance of the *NeedSpecification* activity

This activity instance then is executed; invoked the execution of the program captured in its process component *NeedSpecificationEditor* (Fig. 11).

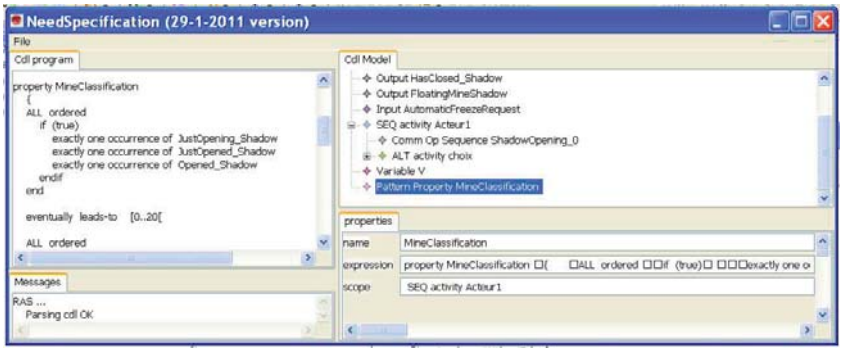


Fig. 11. *NeedSpecificationEditor* process component's screen

When the process component execution is terminated, the OCL constraint *"self.topActivity->size()>0"* is verified and determines the next state of the *NeedSpecification* activity.

5 Conclusion

We have presented our work on supporting software process enactment, concretely to provide an execution environment to MODAL process models. Our approach focuses on integrated supports for controlling the performance of process' activities, i.e. to closely manage used tools and manipulated products in order to satisfy the process' intention.

Closest to our work are the extensions of SPEM 2.0 done in xSPEM[2] and eSPEM[5]. xSPEM adds to SPEM 2.0 the features to model characteristics of a project and to store process's states during enactment then proposes to validate the process model by translating it to a Petri net or to monitor the project by mapping the process model to BPEL code. eSPEM substitutes the SPEM 2.0 behavior interfacing concepts by more fine-grained concepts based on the UML behavior modeling concepts and uses state machines to model process lifecycle. Both of these works do not handle the verification of process intention achievement.

A prototype of the MODAL process execution environment has been implemented, which is able to model, instantiate and simulate MODAL-based process models. The process execution environment has been validated by executing exemplary software process models. Currently, we are investigating a method to capture and manage snapshots of process instances to enable the analysis and possibly the process mining.

6 References

[1] Bendraou, R., Sadovykh, A., Gervais, M.-P. 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.

[2] Bendraou, R., Combemale, B., Cregut, X., Gervais, M.P.: Definition of an executable SPEM 2.0. In: Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC 2007), Washington, DC, USA, pp. 390–397.

[3] Bendraou, R., Jezéquél, J-M., Gervais, M.P. and Blanc, X. "A Comparison of Six UML-Based Languages for Software Process Modeling", in IEEE Transactions on Software Engineering, 2010

[4] 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.

[5] Ellner, R.; Al-Hilank, S.; Drexler, J.; Jung, M.; Kips, D.; Philippsen, M.: eSPEM - A SPEM Extension for Enactable Behavior Modeling. In Proceeding of European Conference on Modelling Foundations and Applications (ECMFA 2010), pp. 189-200 .

[6] Issa Dallo, J. Champeau. An approach for describing concurrency and Communication of heterogeneous systems. First Topcased Days Toulouse 2011

[7] Koudri, A., Champeau, J.: MODAL: A SPEM Extension to Improve Co-design Process Models. New Modeling Concepts for Today's Software Processes (ICSP 2010

[8] Koudri, A., Champeau, J., Le Lann, J.C. and Leilde, V. MoPCoM Methodology: Focus on Models of Computation. Modelling Foundations and Applications (ECMFA 2010

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

[10] OMG, SPEM2.0, "Software & Systems Process Engineering Meta-Model Specification", at http://www.omg.org/cgi-bin/doc?formal/08-01-04/, 2008.

[11] OMG: Uml profile for Marte, beta 1. Technical Report ptc/07-08-04, 2007

[12] OMG: Business Process Model and Notation (BPMN) 2.0, January 2011

[13] P. Dhaussy, P.-Y. Pillain, S. Cre_, A. Raji, Y. L. Traon, and B. Baudry. Evaluating context descriptions and property definition patterns for software formal verification. Model Driven Engineering Languages and Systems (MoDELS 2009).

Component-oriented Multi-metamodel Process Modeling Framework (CoMProM)

Fahad R. Golra and Fabien Dagnat

Université Européenne de Bretagne
Institut Télécom / Télécom Bretagne
Brest, France
{fahad.golra,fabien.dagnat}@telecom-bretagne.eu
http://www.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. The automation of software process modeling is indispensable to keep up with the pace of industrial development needs. The use of component concepts also needs to be exploited for the an optimum support for process improvement. Component-oriented multi-metamodel software process framework (CoMProM) is presented in order to automate the software development processes allowing the flexibility to take processes as components.

Keywords: Process, Metamodel, Multi-metamodel development, Component, Transformation

1 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 [3]. The progress towards the achievement of this goal needs corresponding process modeling support, which has been quite overlooked [14].

The factors restraining software industry to unleash the full potential of Model Driven Engineering are quite explored [9]. 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 [1]. 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.

2 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 [12]. 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 [11]. 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 interoperation mechanism between agents. 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 [10]. 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 [5]. 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 reactive control. This approach is mapped to XPDL for its execution. Though MDE is used for the specification of this approach but it still lacks a concrete support for model driven software development.

xSPEM stands out amongst all the process modeling languages discussed earlier in terms of execution semantics [4]. This approach extends SPEM with project management capabilities, a better process observation and event descriptions. This gives a basis for the process execution but still relies on mappings to BPEL for 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 Petrinet model.

Though software process modeling languages are quite studied and researched, BPMN [13] still remains as the preferred technology for use in industry. The reason for its wide usage is its simplicity, standardization, and its support for execution of processes. Initially BPML [6] took process as a set of activities executed in a context, where each activity has its own properties like local variables, exceptions, and internal functions. A process was triggered from either an event or an explicit call from another process or activity. BPML being neither easily understandable nor graphical had to be assisted by BPMN [13], which focused the dynamic aspects of processes and presented a graphical notation for them. Later on, BPMI dropped support for BPML in favor of WS4BPEL [2]. BPMN standard does not specify a mapping to WS4BPEL for execution but such a mapping is easily deducible and thus is used frequently in the industry. Any specialized software process modeling language has not been able to replace the general business process modeling approaches.

All software process modeling 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. Majority of these approaches rely on BPEL and XPDL for their execution. 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.

3 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.

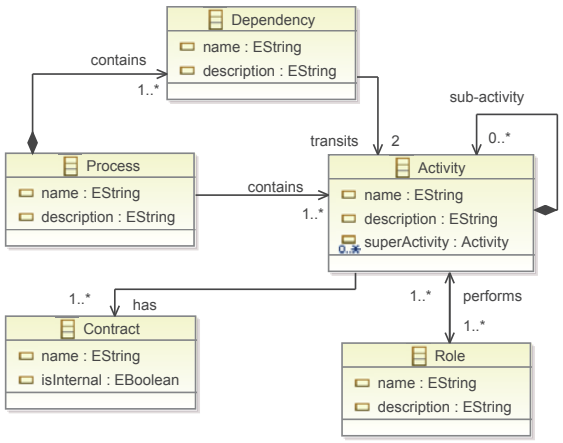


Fig. 1. Core Package

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 sequence 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 be either 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

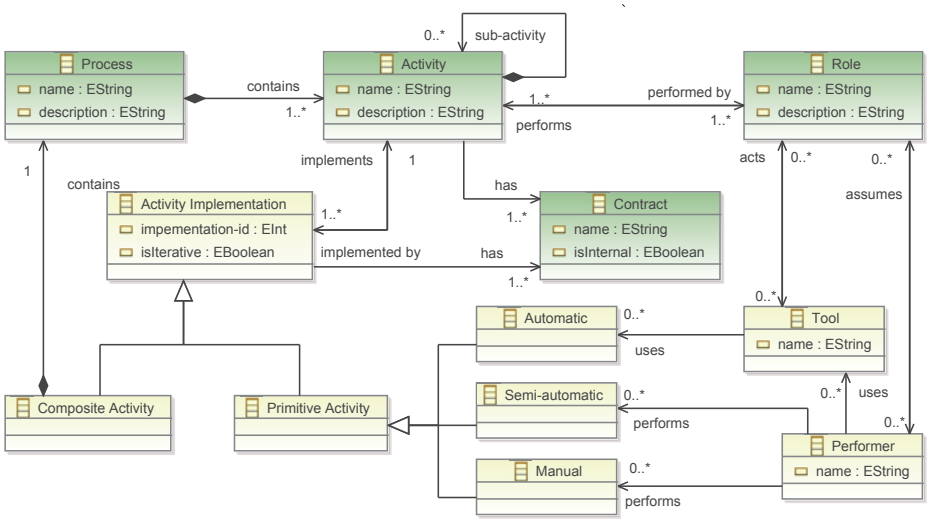


Fig. 2. Activity Implementation Package

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 producable 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 producable 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

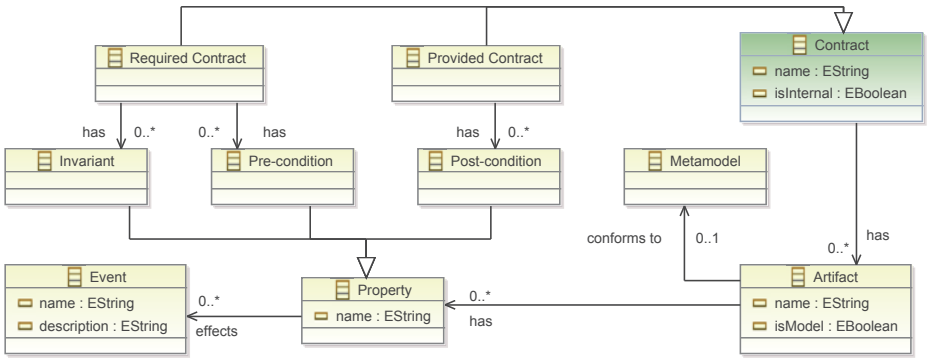


Fig. 3. Contract Package

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 performance 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 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.

4 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 the project, thus it takes into account the time plan and status of the project. Model transformations are used amongst the models conforming to these metamodels. This global picture is illustrated in Figure 4.

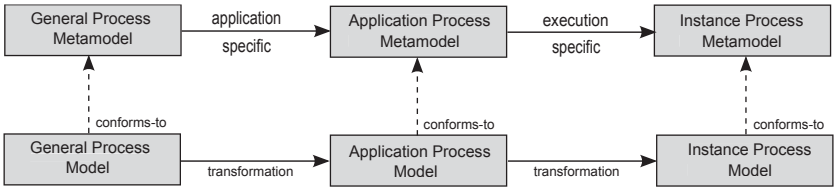


Fig. 4. Process metamodels for Multi-metamodel development

The Component-oriented Multi-metamodel Software development Process framework (CoMProM) is presented through these three different metamodels. For the reasons of brevity, we have only presented the first metamodel i.e. General Process metamodel. The Application Process Metamodel refines this metamodel with some additional features so as to keep it application specific. 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 statistics can be compared and analyzed against the expected statistics. This gives the ability to use the reactive control for process sequencing to reschedule the processes accordingly.

In our approach, we use model transformations amongst these metamodels. The vision of this software process modeling framework is that an organization

can keep their 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 [8]. 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.

5 Conclusion

CoMProM framework presents activities as process components which have their defined contracts. The overall hierarchical structure of the framework allows to model processes at different abstraction levels. Moreover, the framework is itself defined using three metamodels i.e. General, Application and Instance metamodel, 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 the use of familiar jargon for the domain experts.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business Process Management: A Survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) Business Process Management. Lecture Notes in Computer Science, vol. 2678, pp. 1–12. Springer (2003)
2. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services, version 1.1 (May 2003)
3. Ardagna, D., Ghezzi, C., Mirandola, R.: Rethinking the use of models in Software Architecture. In: Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures. pp. 1–27. QoSA '08, Springer-Verlag, Berlin, Heidelberg (2008)
4. Bendraou, R., Combemale, B., Crégut, X., Gervais, M.P.: Definition of an eXecutable SPEM2.0. In: 14th Asian-Pacific Software Engineering Conference (APSEC). pp. 390–397. IEEE Computer Society, Nagoya, Japan (Dec 2007)
5. Bendraou, R., Gervais, M.P., Blanc, X.: UML4SPM: An Executable Software Process Modeling Language providing high-level abstractions. Enterprise Distributed Object Computing Conference, IEEE International 0, 297–306 (2006)

6. BPMI: Business Process Modeling Language. Business Process Modeling Institute, Version 1.0 (November 2002)
7. Chou, S.C.: A Process Modeling Language consisting of high level UML-based diagrams and low level Process Language. Journal of Object Technology 1(4), 137–163 (Sep 2002)
8. Englebert, V., Heymans, P.: Towards More Extensible MetaCASE Tools. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 4495, pp. 454–468. Springer Berlin / Heidelberg (2007)
9. Forward, A., Lethbridge, T.C.: 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. pp. 27–32. MiSE '08, ACM, New York, NY, USA (2008)
10. Koudri, A., Champeau, J.: MODAL: A SPEM Extension to improve Co-design Process Models. In: New Modeling Concepts for Today's Software Processes, Lecture Notes in Computer Science, vol. 6195, pp. 248–259. Springer Berlin / Heidelberg (2010)
11. Nitto, E.D., Lavazza, L., Schiavoni, M., Tracanella, E., Trombetta, M.: Deriving executable process descriptions from UML. In: ICSE '02: Proceedings of the 24th International Conference on Software Engineering. pp. 155–165. ACM, New York, NY, USA (2002)
12. OMG: Software and Systems Process Engineering Metamodel Specification. Version 2.0 (April 2008), `http://www.omg.org/spec/SPEM/2.0/`
13. OMG: Business Process Modeling Notation (BPMN). Version 2.0 (January 2011), `http://www.omg.org/spec/BPMN/`
14. Van Der Straeten, R., Mens, T., Van Baelen, S.: Challenges in Model-Driven Software Engineering. In: Models in Software Engineering, Lecture Notes in Computer Science, vol. 5421, pp. 35–47. Springer Berlin / Heidelberg (2009)