An Integrated Tool Chain for Software Process Modeling and Execution

Ralf Ellner[2], Samir Al-Hilank[2], Martin Jung[2],
Detlef Kips[1,2], and Michael Philippsen[1]

[1] University of Erlangen-Nuremberg, Computer Science Department,
Programming Systems Group, Martensstr. 3, 91058 Erlangen, Germany
philippsen@cs.fau.de
[2] develop group Basys GmbH, Am Weichselgarten 4, 91058 Erlangen, Germany
ellner|alhilank|jung|kips@develop-group.de

Abstract. The Eclipse Process Framework (EPF) allows for a detailed modeling of software development processes and methods based on the Software and Systems Process Engineering Metamodel (SPEM) standard. A comprehensive electronic process guide may be generated from such a model. However, EPF and SPEM only support a rather coarse description of the behavior of software processes. As there is no support for automated enactment or simulation of these software process models, one cannot benefit from context-sensitive process guidance, automated process conformance checking, or automated progress tracking when enacting detailed software process models.

eSPEM (enactable SPEM) is an extension of the SPEM standard that supports UML activities and state machines for fine-grained behavior modeling. Its operational semantics is based on OMG's fUML (Semantics of a Foundational Subset for Executable UML Models) and may be used to instantiate, simulate, and enact software process models. However, without a reasonable tooling for eSPEM the benefit for end users is still limited.

This paper presents an integrated tool chain based on eSPEM and Eclipse. The tool chain not only supports process modelers in modeling fine-grained eSPEM-based software processes, but also guides and supports project staff in working according to the process in a context-sensitive manner. It automates repetitive and cumbersome work like checking process conformance or progress tracking. Hence, it lets end users benefit from process modeling and enactment.

1 Introduction

Software development processes (SDPs) are widely accepted as a critical factor in the efficient development of complex and high-quality software and systems. Beginning with Osterweil's process programming [1] many process modeling languages (PML) have been proposed to describe SDPs in more or less abstract, (semi-)formal ways, see [2–4] for an overview.

SPEM [5] is a standardized PML; it is based on the UML Infrastructure [6] and defines a graphical notation. Due to its familiar notation, practitioners can pick up SPEM easily. The Eclipse Process Framework (EPF) provides a reference implementation of SPEM. However, SPEM and EPF have been primarily designed to model and document the static structure of SDPs. Thus, when a SDP is modeled with SPEM, this results in a thorough informal documentation of the process. With EPF, one may generate an electronic process guide from a SPEM-based SDP model. Such a process documentation is valuable or may even be required, for example in safety critical projects, but it does not provide much additional value for the project staff as there is no help in executing the process.

Although it has been a requirement, executability is not in the scope of the current version 2.0 of SPEM, even though it would provide the following additional benefits (see [1, 7]): First, executable software process models can be simulated and can thus more easily be validated before they are used in a project. Second, a process execution machine (PEX) can guide and support the project staff. Third, since a PEX can automatically check conformance of the executed process with the modeled process, it can detect and prevent process violations. Finally, a PEX can track progress of the executed process. This is of great use for process audits, because it is possible to partially automate the proof that the actually executed process conforms to the modeled process.

In [8] we presented eSPEM, a SPEM extension based on UML activities and state machines [9]. In addition to the behavior modeling concepts of UML (for example, decisions, exceptions, and events), eSPEM also provides behavior modeling concepts that are specific to SDPs (e.g., task scheduling). These behavior modeling concepts can be used to describe the behavior of SDPs in a fine-grained, formal, but intuitive way. The formality of the SDP behavior description is required in order to execute it. Another requisite of SDP execution is that there is a rigid definition of the operational semantics used to describe a SDP. In [10] we presented the operational semantics of eSPEM based on the OMG standard *Semantics of a Foundational Subset for Executable UML Models* (fUML) [11]. fUML defines the operational semantics of UML activities and actions. But fUML execution is limited to a single machine, and fUML does not support human interaction nor can its execution model be extended. While fUML is suitable for local simulation, it is insufficient for distributed process execution which is needed for typical team-driven software projects. Thus, we added support for distributed execution, human interaction, and user specific extensions to the operational semantics of eSPEM. We also implemented the operational semantics of UML state machines which are missing in fUML.

Our extensions to SPEM and fUML are the conceptual foundation of a detailed software process modeling and automated enactment. However, a reasonable tool support for eSPEM is required to let end users benefit. This papers presents such an integrated tool chain for software process modeling and execution. Fig. 1 gives an overview of the tool chain from a user's perspective.

A *Process Designer* models, validates, and simulates an executable software process using a *Process Modeling Environment (PME)*. The *Process Designer*

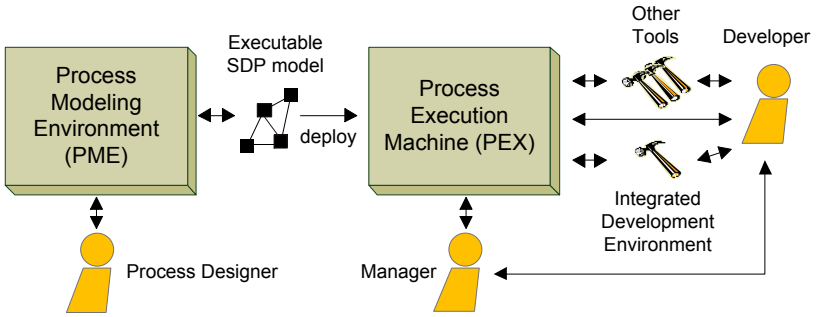

Fig. 1. Overview of the Integrated Tool Chain

may then deploy the resulting model to a *PEX*. A *Manager* may use the PEX to create development projects that follow the model. The PEX interprets the software process model and guides and supports the *Developer*s through the process. Depending on a task a developer performs, the PEX offers context sensitive help. The PEX also cooperates with tools used in the process and manages artifacts as specified in the process model. It can report deviations between the modeled process and the process actually executed. In addition, the PEX tracks all actions performed throughout the process and supports process traceability.

Section 2 presents the overall architecture of our integrated tool chain and shows how it achieves the mentioned benefits. In section 3 we use the tool chain to model and enact an exemplary process. Section 4 covers related approaches and tools. In section 5 we conclude and preview future work.

2 Architecture of the Integrated Tool Chain

The tool chain is built from three major components (see Fig. 2). First, a Process Modeling Environment (PME) to model, simulate and deploy software processes, a Process Enactment Server (PES) that manages deployed process models and distributed process model instances, and a Process Enactment Client (PEC) that is the graphical front end for process enactment. PEC and PES employ a traditional client/server architecture, together providing the functionality of a PEX (see Fig. 1). All three components are based on the Eclipse Equinox OSGi framework [12] and several other Eclipse components [13] (see Fig. 2) to help distributed development teams in their process enactment.

2.1 Process Modeling Environment (PME)

The PME implements the abstract and concrete syntax of the modeling language eSPEM. The abstract syntax is realized as an EMF (Eclipse Modeling Framework) Ecore meta-model [14] accompanied by mostly generated Java code. Since typical

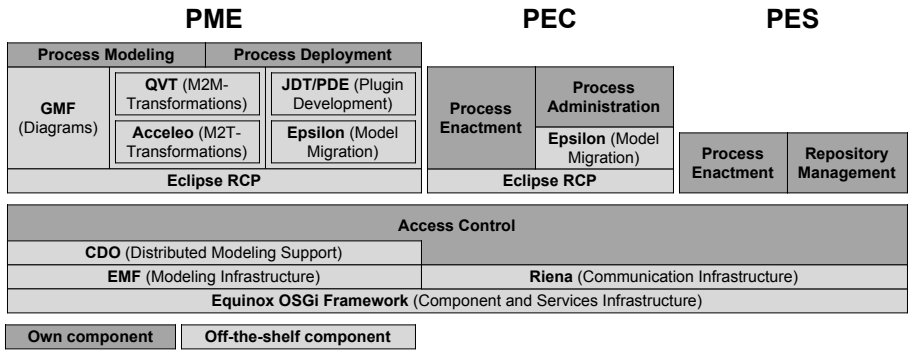

Fig. 2. Overall Architecture of the Integrated Tool Chain

SDP models tend to be large and complex, the PME strives to reduce that complexity. First, a view-based diagram editing approach for eSPEM's concrete syntax makes modeling of processes more intuitive and easier. Four different types of diagrams depict different aspects of the SDP model. There are a state machine diagram editor and an activity diagram editor for the dynamic behavior of software processes. Aspects of the static structure of software processes can be modeled with a method content diagram editor and a team profile diagram editor. All diagram editors are realized with GMF (Graphical Modeling Framework) [15] and may be used concurrently on a single process model.

In order to further reduce complexity of process models and to provide fast and easy access to frequently used information, we also provide a variety of Eclipse views that filter the model by the type of model elements. For example, the activity view shows all activities within a model. These views also show the semantic relations among model elements, e.g., which roles perform an activity. As a result, these views provide fast access to a model element and its usage within the model. In addition, there is an electronic process guide (EPG) preview that provides a comprehensive, navigable electronic process documentation in natural language. It uses a model-to-text (M2T) transformation to generate an EPG and to display it in a browser.

For general purpose modeling, the model explorer provides an unfiltered view of the abstract syntax tree of a process model. It can be used to generically create, select, move, and delete elements. In addition, model validation can be triggered from the model explorer. Our validation support is based on the EMF Validation Framework [14] and various OCL [16] constraints in the eSPEM meta-model. This automated validation support is a great help in checking a large model for structural and semantic correctness. For even more validation support, there is the process simulation view. It may be used to locally simulate the behavior of a process on top of the eSPEM execution machine. Process simulation greatly improves debugging and testing SDPs and finding bottlenecks in them.

As most software processes have to be tailored to the needs of a concrete project, eSPEM provides language constructs to adapt and select processes and development methods, and to bring them together as a process configuration [8, 5]. To ease process configuration, the configuration view helps to bind a process to methods by simple drag-and-drop.

Similar to process tailoring, a PEX must be adaptable to the needs of a concrete project or organization. For this kind of adaption the tool chain provides a lot of extension points for plugins. Plugins may be developed with the PME that also includes all Eclipse components for Java and Eclipse plugin development. There are extension points for tool adapters, version control systems, extensions of our process runtime infrastructure, and process plugins. To deploy and enact a process model, a corresponding process plugin for our PES/PEC has to be generated first. This task is fully automated. Model-to-model and M2T transformations turn an eSPEM-based process model into an Ecore model, Java code, and meta data (e.g., OSGi meta data, EPG) that form a process plugin. Finally, process plugins may adapt an already running process to a new version of its process model. For this purpose, we use Epsilon Flock [17], a transformation language for model migrations. Flock greatly simplifies model migrations because only rules for transforming structural differences have to be written explicitly at the level of the generated Ecore models. Unchanged parts of two meta-models are adapted automatically.

2.2 Process Enactment Server (PES)

The PES provides secure shared access to process instances. It manages eSPEM execution models with CDO [13], a framework that allows for concurrent remote access to models and supports model versioning. We use the latter to implement traceability of process enactment. This is useful in process audits and for process improvements. Since versioning leads to large amounts of data, we use a relational database as back-end to store execution models. To secure access to process instances, we implemented an access control layer on top of CDO and Riena [18].

The eSPEM process execution machine relies on the observer design pattern to track changes of the execution model. Upon each change, the next execution steps are computed and performed. For performance reasons, the PES manages the observer object that has direct access to process instances in the Java virtual machine of the PES.

2.3 Process Enactment Client (PEC)

The PEC provides views and editors to instantiate, manage, and execute deployed software process models and their instances. It is an Eclipse rich client application and may be directly integrated into the development environment used by the project staff. The PEC presents two predefined perspectives: First, the administration perspective with views and editors to create and administrate projects, users, and plugins. Second, the process enactment perspective that the project staff uses to execute the process, i.e., manage activities, tasks, and work

products. Context-sensitive help from the electronic process guide is seamlessly integrated into those perspectives to allow a direct access to the relevant part of the process description.

Process audits often require a thorough documentation of each process step. Without an automated enactment this results in a lot of additional manual labor. The PEC/PES fully automates this work and automatically generates log messages that make all changes to the process state reproducible. The PEC's process log view is another benefit as it provides easy access to the generated log messages.

Due to the complexity of software processes, it is often a nontrivial task to follow them. It is even more difficult to fix conformance problems after things have gone wrong. eSPEM uses (OCL) constraints to detect such problems. Checking of conformance is bundled with a transaction system that performs rollbacks in case of severe problems or issues warnings in case of less severe problems. This automated process instance validation and rollback helps project staff in working according to the process by reducing time to detect problems.

3 Walkthrough of an Exemplary SDP

To demonstrate the functionality of our integrated tool chain, we first cover the Scrum SDP [19] as an exemplary process. Scrum is an agile, iterative-incremental process. An iteration is called *sprint*. Within each sprint, a potentially shippable product increment is developed. A sprint starts with a *sprint planning meeting*. During this meeting, tasks are estimated and written down in the *sprint backlog*. After the sprint planning meeting, the planned tasks are executed by project staff. Each day starts with a *daily scrum meeting* followed by development work. At the end of a sprint, the developed product increment is presented during the *sprint review meeting*. Afterwards, the sprint is discussed and process improvements are elaborated during the *sprint retrospective meeting*.

3.1 Modeling Scrum

After this sketch of Scrum, we illustrate how to model a part of Scrum with a PME, deploy the model to a PES, and enact Scrum with a PEC. We provide a more detailed version of this example as screencast in [20]. Our tool chain can be downloaded from [21].

Since software development is a creative endeavour, not all details of a process can be anticipated and modeled. In fact, most processes (including Scrum) use dedicated planning activities (e.g., the sprint planning meeting) to react upon changing requirements or risks within a project. However, current process modeling languages have no means to express such process situations. In contrast, eSPEM provides task schedulers to model the planning of dynamically instantiated tasks. Fig. 3 shows how a task scheduler (`Backlog Task Scheduler`) may be used to model an execution strategy (e.g., based on the priority or dependencies of a task) for tasks within the `Sprint Backlog`.

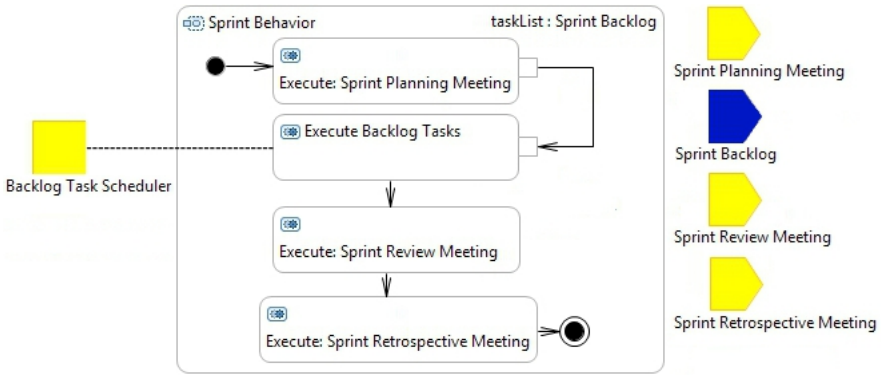

Fig. 3. Behavior model of a Scrum sprint modeled with eSPEM and our PME

The special action `Execute Backlog Tasks` accepts the `Sprint Backlog` and executes its tasks in the order determined by the `Backlog Task Scheduler`. Using this modeling concept, we integrated the main project planning behavior of Scrum into our process model. The two additional actions in Fig. 3 have simple call semantics executing the tasks they refer to (`Sprint Review Meeting` and `Sprint Retrospective Meeting`). In the same way, the behavior of a Sprint shown in Fig. 3 may be called by another activity.

Although this example lacks some detail (see [20] for a more detailed step-by-step presentation), we expressed the main behavior of a Scrum sprint with just a few model elements. This behavior may be simulated with a PME or enacted with a PES to drive a Scrum-based project. For the latter, a user must deploy the process model as a process plugin to a PES.

3.2 Enacting Scrum

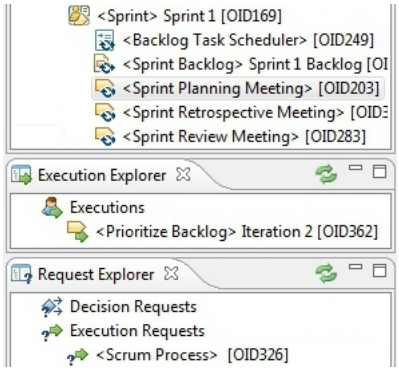

Fig. 4. Scrum Enactment with our PEC

After a process plugin is successfully deployed to a PES, a PEC may be employed to create a development project that uses this process. When a project is created the PES automatically creates an initial activity, artifact repositories, and proxy objects for the process roles and tools. Afterwards, the behavior of the initial activity must be started manually to execute the process. In our example, the `Sprint Behavior` is executed. As shown in Fig. 3, a sprint planning meeting has to be executed first in a sprint. To guide the user, the PES interprets the process model and

creates a request to execute a sprint planning meeting. A PEC displays these pending execution requests in a dedicated view called `Execution Explorer` (see Fig. 4). This view provides context-sensitive actions to mark a task as finished. When a user marks the sprint planning meeting as finished, the PES executes the next action (`Execute Backlog Tasks`) by inspecting the `Sprint Backlog` (provided as input parameter) and scheduling the tasks in the backlog according to the `Backlog Task Scheduler`. This results in further execution requests in the `Execution Explorer` that must be handled by a user. When all requests are handled, execution of a `Sprint Review Meeting` and a `Sprint Retrospective Meeting` is requested by the PES. Finally, the control flow returns to the caller of the `Sprint Behavior` that may start another sprint. With this short example, we demonstrated how to model a SDP with our PME, deploy it, and enact it with a PES and a PEC.

4 Related Work

Many authors have identified software process modeling and execution as relevant for producing software. Early approaches of software processes modeling are executable [2] but had limited impact in industry due to their complex formalisms, low level of abstraction, or inferior tool support [22]. Therefore, below we focus on high-level modeling languages with up-to-date tool support.

The Microsoft Team Foundation Server (TFS) [23], IBM Rational Team Concert (RTC) [24], and Method Park Stages [25] are popular Application Lifecycle Management (ALM) tools that integrate other tools, e.g., configuration management systems, change management systems, and IDEs into a distributed development and collaboration platform. Although they can be adapted to a process by using templates, their template languages are limited to static aspects of the processes only. In contrast to our work, these tools cannot guide and support project staff in executing a custom process based on a process template, e.g., they cannot determine the next possible steps in a process and guide project staff accordingly.

UML4SPM [26] extends SPEM 1.1 with UML 2.0 behavior modeling concepts. Although its operational semantics has been implemented on top of fUML to simulate and execute UML4SPM-based process models [27], there is no support for distributed execution that is needed for realistic team-based development.

The Eclipse Process Framework (EPF) [28] offers a reference implementation of SPEM 2.0. With the IBM Rational Method Composer [29] a commercial version of EPF is also available. Although comprehensive EPGs may be generated from such EPF models, they lack a description of the behavior as SPEM lacks suitable language constructs. Hence, EPF cannot simulate or enact a process and thus cannot help exploit the mentioned benefits, e.g., it makes process validation more complex.

The Process Enactment Toolkit (PET) [30] is a framework to enact process models. PET is a generic model-to-model transformation framework with input adapters for different process modeling languages and output adapters for different

issue management, collaboration, and ALM tools like TFS. As stated above, TFS and other ALM tools fall short in guiding and supporting project staff. Moreover, in contrast to our work PET cannot simulate process execution.

5 Conclusion and Future Work

In this paper, we presented an integrated tool chain for modeling and enacting software processes. The Eclipse-based tool provides a reasonable support for modeling, documenting, simulating, and enacting eSPEM-based software processes. It also comes with a process simulator to easily validate a process before it is enacted. Our context-sensitive help provides easy access to the relevant part of the process documentation. Our automated process logging keeps accurate data for process audits and improvements without any additional effort for project staff. Furthermore, the tool automatically checks for process conformance and proactively prevents illegal process states. Our tool chain relies on the Eclipse architecture to integrate other Eclipse-based extensions and third party tools. Currently no other tool provides such a completely integrated approach that leverages all mentioned benefits of automated process enactment.

Our future work will extend the scope of our integrated tool chain to support process assessments and to check conformance of process models to process reference models, e.g., CMMI [31], and standards like ISO 26262 (Road vehicles – Functional safety) [32]. Since our tool chain provides access to the process model and runtime data, it is an ideal environment to check conformance of processes. We integrate standard models and trace models with our tool chain in order to support assessments of process models, and assessments of executed processes in multi-certification contexts (i.e., processes must conform to more than one standard).

References

1. Osterweil, L.: Software processes are software too. In: Proc. 9th Intl. Conf. Software Eng., Monterey, CA. (Apr. 1987) 2–13
2. Zamli, K., Lee, P.: Taxonomy of Process Modeling Languages. In: Proc. ACS/IEEE Intl. Conf. Computer Sys. and Appl., Beirut, Lebanon. (Jun. 2001) 435–437
3. Acuña, S.T., Ferré, X.: Software Process Modelling. In: Proc. World Multiconf. Systemics, Cybernetics, and Informatics, Orlando, FL. (Jul. 2001) 237–242
4. Bendraou, R., Jezequel, J.M., Gervais, M.P., Blanc, X.: A Comparison of Six UML-Based Languages for Software Process Modeling. IEEE Trans. Softw. Eng. **36**(5) (2010) 662–675
5. Object Management Group: Software & Systems Process Engineering Meta-Model Specification, Ver. 2.0. (Apr. 2008)
6. Object Management Group: OMG Unified Modeling Language, Infrastructure, Ver. 2.3. (May 2010)
7. Almeida da Silva, M., Bendraou, R., Blanc, X., Gervais, M.P.: Early Deviation Detection in Modeling Activities of MDE Processes. In: Proc. 13th Intl. Conf. Model Driven Eng. Lang. and Sys., Oslo, Norway. Volume 6395 of LNCS. (2010) 303–317

8. Ellner, R., Al-Hilank, S., Drexler, J., Jung, M., Kips, D., Philippsen, M.: eSPEM - A SPEM Extension for Enactable Behavior Modeling. In: Proc. 6th Europ. Conf. Model. Foundations and Appl., Paris, France. Volume 6138 of LNCS. (2010) 116–131
9. Object Management Group: OMG Unified Modeling Language, Superstructure, Ver. 2.3. (May 2010)
10. Ellner, R., Al-Hilank, S., Drexler, J., Jung, M., Kips, D., Philippsen, M.: A FUML-based Distributed Execution Machine for Enacting Software Process Models. In: Proc. 7th Europ. Conf. Model. Foundations and Appl., Birmingham, UK. Volume 6698 of LNCS. (2011) 19–34
11. Object Management Group: Semantics of a Foundational Subset for Executable UML Models, Ver. 1.0 Beta 3. (Mar. 2010)
12. McAffer, J., VanderLei, P., Archer, S.: OSGi and Equinox. Addison-Wesley Longman (2010)
13. Eclipse Project: `http://eclipse.org/`. (Apr. 2012)
14. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. 2nd edn. Addison-Wesley Longman (2009)
15. Gronback, R.C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Longman (2009)
16. Object Management Group: Object Constraint Language, Ver. 2.2. (Feb. 2010)
17. Rose, L., Kolovos, D., Paige, R., Polack, F.: Model Migration with Epsilon Flock. In: Theory and Practice of Model Transform. Volume 6142 of LNCS. (2010) 184–198
18. Riena Platform Project: `http://eclipse.org/riena/`. (Apr. 2012)
19. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press (2004)
20. Al-Hilank, S., Ellner, R.: Modeling and Enacting Scrum (Screencast), `http://www2.cs.fau.de/research/IWKMMASWEP/screencasts/`. (Apr. 2012)
21. Al-Hilank, S., Ellner, R.: eSPEM and tool chain download page, `http://www2.cs.fau.de/research/IWKMMASWEP/download/`. (Apr. 2012)
22. Gruhn, V.: Process Centered Software Engineering Environments – A Brief History and Future Challenges. Annals of Softw. Eng. **14**(1-4) (2002) 363–382
23. Microsoft: VS Team Foundation Server, `http://microsoft.com/vs/`. (Apr. 2012)
24. IBM: Rational Team Concert, `http://ibm.com/rational/rtc/`. (Apr. 2012)
25. Method Park Software AG: Stages, `http://methodpark.com/en/product.html`. (Apr. 2012)
26. Bendraou, R., Gervais, M.P., Blanc, X.: UML4SPM: A UML2.0-Based Metamodel for Software Process Modelling. In: Proc. 8th Intl. Conf. Model Driven Eng. Lang. and Sys., Montego Bay, Jamaica. Volume 3713 of LNCS. (2005) 17–38
27. Bendraou, R., Jezéquél, J.M., Fleurey, F.: Achieving process modeling and execution through the combination of aspect and model-driven engineering approaches. J. of Softw. Maintenance and Evolution: Research & Practice (2010) Preprint.
28. Eclipse Process Framework Project: `http://eclipse.org/epf/`. (Apr. 2012)
29. IBM: Rational Method Composer, `http://ibm.com/rational/rmc/`. (Apr. 2012)
30. Kuhrmann, M., Kalus, G.: Providing Integrated Development Processes for Distributed Development Environments. In: Workshop on Supporting Distributed Team Work at Computer Supported Cooperative Work (CSCW 2008). (Nov. 2008)
31. CMMI Product Team: CMMI for Development, Ver. 1.3. Technical Report CMU/SEI-2010-TR-033, Carnegie Mellon Univ. – Software Eng. Inst. (Nov. 2010)
32. Intl. Org. for Standardization: ISO 26262: Road vehicles – Functional safety. (Nov. 2011)