The Process Enactment Tool Framework - Transformation of Software Process Models to Prepare Enactment

Marco Kuhrmann¹, Georg Kalus, Manuel Then

Technische Universität München, 
Faculty of Informatics – Software & Systems Engineering, 
Boltzmannstr. 3 
85748 Garching, Germany

¹Corresponding Contact: 
E-Mail: kuhrmann@in.tum.de 
Phone: +49 89 289 17386

The Process Enactment Tool Framework –
Transformation of Software Process Models to Prepare Enactment

M. Kuhrmann, G. Kalus, M. Then

Technische Universität München, Institut für Informatik – Software & Systems Engineering, Boltzmannstr. 3, 85748 Garching, Germany

Abstract

Rich development process models contain information about structures for project organization and also for concrete outcomes of a project. However, rich processes are hard to implement. They often contain hundreds of pages of documentation. Development teams tend to be sceptical about rich processes in fear of additional effort, risking the benefits of rich tool support for enactment. Process enactment is a challenging task. There is no common methodology to quickly ”implement” a development process in a tool or a set of tools. Often specialized tools are used to provide assistance during the project and it is the project manager’s task to consolidate the information with the rest of the team.

The Process Enactment Tool Framework (PET) is a software tool that supports the transformation of a given formal development process into a format that project tools can work with. PET is an instrument to import processes based on a metamodel and provide exports for a specific project environment. PET takes an input software development process model and transforms it into an intermediate format that serves as the basis for a second transformation step into data formats of tools such as office suites or comprehensive ALM platforms. In this paper we present the tool framework and show how metamodel-based processes can be transformed into an environment that is ready to use for a project team. We show how PET is applied for the German V-Modell XT and for SPEM-based processes to generate, e.g., process templates for the Team Foundation Server or work product document templates.

Keywords: development process models, process model mapping, enactment, project initialization support

1. Introduction

"Nice work with the process, but how does it support me during the project?" – it is such questions that we are facing in our day-to-day work with partners from industrial practice. The benefit and advantages of following a defined process is sometimes not immediately evident to the project team [43]. A development process is a tool that supports the cooperative work of development teams by providing the terminology to be used, knowledge about practices, techniques,
and methods that are considered useful in a project. Furthermore a development process supports
the management of a project to organize the team, assign work packages and responsibilities, and
so on.

A problem with most development processes, especially with the ones that are referred to as
"heavy", rich, or plan-driven processes is that they are often perceived as basically consisting
of lots of documentation inhibiting creative freedom. Thus, such processes often are not well
accepted. The team is not interested in working on tasks prescribed by the process, which – at first
sight – have nothing to do with the project’s objectives. The implementation of a development
process is therefore either difficult and time consuming or not done at all. The process may
even be “faked” [41]. Development teams tend to prefer agile practices that specifically focus on
some development tasks and rituals accompanying the development. Development tools usually
play an important role in such settings. Tools are selected according to the development tasks,
i.e. programming or testing tools. Rituals, such as the Daily Scrum [46], are embedded in the
day-to-day work process.

Using tools to support the implementation of (development) processes has influence on the
acceptance and the applicability of the process. In the context of rich processes, tools can not
only support the development tasks, but can also be useful to control and enforce the application
of the process and most importantly, tools can ease the application of a development process.

1.1. Problem Statement

Process enactment itself is not a new topic (see Sect. 2). Many organizations face the issue
of having a (defined) process model on the one hand and a set of tools to be used in projects on
the other, but no strategy to systematically bridge the gap between processes and tools (Fig. 1).
A rather common strategy is to simply distribute the process documentation, maybe give some
instructions about how to apply it and sometimes audit at a later stage to see whether the project
is operated according to the defined process. Opportunities given by rich and structured process
models often remain unused:

1. Rich process models define artifact types for manifold project objectives, either workflows,
code- or document templates. Taking into account that rich process models are usually
based on metamodels, tools can make use of such well-defined structures to build up a
project data model.

2. Rich process models usually define comprehensive processes that can serve as a basis for
project execution (e.g., ordered milestones with assigned artifacts). Tools can make use of
such information, i.e. to derive project plans.

3. Rich process models often define dependencies between artifacts. Software tools can make
use of that information, e.g., to generate checklists for quality assurance, measure project
progress, and so on.

Current application life cycle (ALM) platforms, such as IBM Jazz [25] or Microsoft Team
Foundation Server [36], are powerful in terms of direct enactment (e.g., by implementing micro
processes or work items using a work flow engine, cf. Sect. 2) but hardly connect to comprehen-
sive process models.

Challenges for process enactment in general and PET in particular are the utilization of one
selected development process in an environment consisting of various tools as well as to support
multiple process models with the same tool. This implies individual mappings between processes
and tools, their combination to a mapping of one process and a set of tools and finally to a
setting, where many processes can be enacted by many tools. Such a concept needs to respect dependencies between processes and tools. The goal of PET is to close the gap between defined, metamodel-based processes and existing tools for project operation.

1.2. Contribution

In this paper we present the Process Enactment Tool Framework (PET), a software tool that allows the instantiation of process models in project-supporting tools. PET makes use of the well-defined structures of metamodel-based process models, converts those structures into a tool-independent intermediate model, and creates templates and/or comprehensive environments for certain tools (Fig. 1). PET is a low-level, technical solution to reorganize process models in a manner that project tools can interpret and enact. Due to the design of PET, no changes to the target tools have to be made. PET is an academic platform to elaborate basic questions regarding enactment. Some standard use cases for selected process-tool combinations have reached a "product-ready" maturity. We are going to present some of the typical use cases later in the paper.

What PET is not meant to be. PET is a software tool that only supports the transfer from process models into tools commonly used in projects. PET is not a process modeling tool such as EPF [17]. The tool rather uses processes that have been defined already (for example using EPF).

The intermediate model, which PET uses to map process models and tool data/execution models, is not meant to be yet another general process metamodel. It is just used internally to decouple input process model and the data format required by the target tool.

PET is also not meant to be an alternative for approaches such as xSPEM [6], which enriches the SPEM metamodel [38] by "executable elements". PET may however benefit from xSPEM as it potentially opens the doors for better tool support in a development project (a more detailed discussion on this topic can be found in Sect. 2).
1.3. Outline

The remainder of this paper is organized as follows: Section 2 contains a discussion of the related work and a description of the tool’s development history. Section 3 introduces the architecture of PET. The section covers the core components of PET as well as the basic transformation process and the plugin concept. In Sect. 4 we introduce a couple of the existing interfaces to process models and tools. In Sect. 5 we give some information of PET’s application. The sections 6 and 7 discuss the challenges during the development of PET and the roadmap for the further development. We conclude this paper in Sect. 8.

2. Related Work

In this section we discuss the related work with regards to the Process Enactment Tool Framework. In recent years, several contributions with respect to process enactment have been made, ranging from formal modeling aspects, e.g., [1], over proposals for making a process model “executable” in general [6], to concrete tools, such as [36]. A general approach for mapping given process models to a concrete tool (chain) however seems to be missing. While process-supporting tools are no silver bullet for successful (distributed) projects and certainly cannot (and are not meant to) replace qualification, face-to-face communication, etc., evidence suggests that a (distributed) project not following a well-defined process and not using tools to facilitate communication and coordination is much more likely to not be successful [45]. Tool support in general and different kinds of it are discussed in [29]. In this section we cover the most important topics of (direct) process enactment, namely enactment (in general), tools, processes and tools.

Enactment. The implementation of a development process (model) in a concrete project is often called enactment. Friedrich and Bergner [18] provide definitions of enactment while discussing the enactment as named in SPEM 2.0 [38]. Using the SPEM definitions they distinguish enactment to be either direct or plan-based. In this terminology, PET is a tool to enable or support direct enactment. Another approach for enacting process models is given by xSPEM [6]. xSPEM enriches the SPEM metamodel [38] to enable monitoring and tracing of process elements during (project) execution. While xSPEM may lay the foundation for “executable” software development process models as far as the model itself is concerned, it does not specify how such a model would be “brought to life” in a concrete tool during project execution. We are not suggesting that it should. xSPEM is a good approach to make “executable” development process models. The interpretation of such a model and how it is represented in a particular tool is where ideas such as PET come into play.

Tools. Process-based tool support can comprise simple tasks that are generated from the process structure (e.g., [11, 16]), comprehensive schedules [8, 18], templates, or complete working environments (e.g., [14] or [36]). To enact a process, appropriate process elements (artifacts) have to be identified. Moreover it is usually not helpful to give hundreds of pages process documentation to the team, be it pages of a printed book or web pages. Support for the team working with a process often has very little to do with documentation. Project teams rather need specific support for relevant tasks and artifacts. The enactment of a development process depends on the concrete project environments. A lot of different tools can be found for the support of project activities and enactment of development processes. When considering enactment of a software development process by tools, three categories of tools can be distinguished:
1. Specialized tools for selected (development) tasks
2. Tools for supporting the project management
3. Integrated development environments with process execution capabilities – often referred to as application life cycle management tools

Specialized tools can be found mainly in the area of agile methods. They provide small and specific solutions, i.e. unit testing, continuous integration and build tools, task boards, and so on. Additional tools provide comprehensive support, but basically focus on specific (sub) processes or methods, i.e. project- or issue management. Integrated development environments (IDE) or application life cycle management platforms (ALM) provide a platform that may contain several of the above stated components. In addition, IDEs usually have interfaces for the integration of 3rd-party solutions. A difficulty with the tool landscape is that the tools are usually specific for a concrete project or a class of similar projects. If the main business is the development of individual solutions, the tool environment may change from project to project.

Implementation of tools and processes, and tool-support for development processes has been discussed for a couple of years [1, 5, 11, 16, 24]. Most of the (formal) approaches are based on graphs and graph grammars to provide a mechanism for integrating formal processes and tools. Westfechtel for example [20, 4] showed some concepts especially designed for development processes. Most of the formal approaches stayed academic, or were applied just for specific domains, i.e. in the development of embedded systems. Because the embedded system’s domain is comparatively easy to formalize, model-driven development (including code generation, testing, validation and verification of models, etc.) is relatively common in this area. The development of business information systems is less formalized. There is a great variety of techniques for the design (e.g., UML, DSLs, formal methods), development (e.g., various programming languages, Model-driven development), or test (e.g., quality assurance in general and unit testing in particular). Software development processes have to respect this heterogeneity. Although a lot of (commercial) tools are available, process-based tool support is still at the beginning.

Tool support according to development processes became popular and “visible” with agile methods. Especially eXtreme Programming (XP, [3]) was a driver. A lot of small tools, i.e. for unit testing, continuous integration, etc. were developed, and coupled into a tool chain for a concrete project [29]. Tool vendors such as IBM/Rational or Microsoft have picked up this trend and developed integrated tools that offer support for most of the development tasks. Such tools especially aim at mitigating issues of distributed software development by helping teams to collaborate and synchronize, as well as by offering a centralized virtual workspace. Yet, we observe a gap between technically mature development environments and process support. Development and management tend to be regarded as separate and rather independent disciplines. Generic collaboration platforms such as Microsoft SharePoint do not imply a process at all, while tools like IBM Jazz [25], impose a predefined and fairly static process. The Team Foundation Server [36] provides tool support by offering a central data repository and a work item tracking system. The process itself is documented in an separate set of HTML pages that is at best loosely connected to the systems. There are also some small and medium enterprises that provide specialized solutions for particular audiences (e.g., Stages, or microTOOL or Polarion for the German market).

Another research area of interest for our work is the investigation of processes at project-runtime. Project cockpits (Software Project Control Centers, SPCC, [30, 37], or related contributions regarding Software Engineering Environments, e.g., [14]) are used to improve monitoring of a project by providing sophisticated measurement and analysis capabilities. Project cockpits collect heterogeneous data from a project and present them in a way that enables the
management to easily grasp the project’s overall status and trends. Furthermore, if something goes wrong in the project, a project cockpit can provide assistance for example in the form of deescalation strategies, knowledge bases etc. Most currently existing approaches are reactive (passive) cockpits; meaning that they only analyze and give advice. If a cockpit had knowledge of the underlying process, one could imagine it to be proactive to a certain extent. Such a cockpit would not only depend on data provided by reporting engines, but would be controlled by a process. Trends could be identified and corresponding measures could be initiated or suggested to the user. The problem with SPCCs is that most of them are process-agnostic. Metric- and reporting modules can be configured, but no process that maps organization or project structures is provided. SPCCs are basically yet another instrument in the project manager’s tool box, equivalent to project planning. With modern ALM platforms a new perspective on SPCCs was created but there is still a large potential for improvement if the cockpit had “knowledge” of the development process.

Processes and Process Metamodels. The most popular and mature metamodels for software development processes are (SPEM [38]/EPF [17], the V-Modell XT [49] in Germany, and the ISO/IEC 24744 [22]) standard. A process metamodel formalizes the structure and semantics of a concrete process model, according to [21], where it is stated that a (concrete) software development process is an instance of a process (or method) model, which in turn is based on a process metamodel. Current research enhances formalization to make such models more useful in the context of a concrete project, e.g., by providing data models or templates. Contributions can be found in the areas of the process lines approach [44, 47], variant management [27, 48], and semantics [7, 8, 18].

As stated above, tools are available for process integration, but the integration is still at the beginning. The provision of concrete, efficient methods for development processes tailored to a (project-specific) tool infrastructure is of interest [23, 40, 50]. This is especially important as comprehensive and rich standard processes like RUP [26] (as representative for EPF-based process models, e.g., OpenUP [2], or Hermes [13]) or the V-Modell XT [19] are gaining popularity – especially in settings where, e.g., larger organizations have to define an organization-wide basic process, or compliance is a requirement. Those comprehensive process models are well-suited for tool support for the following reasons:

Metamodel: RUP, Hermes, and the V-Modell XT are based on formal metamodels. SPEM [38] is the metamodel for RUP and Hermes, and also for the Eclipse Process Framework (EPF, [17]). A metamodel exist also for the V-Modell XT [49]. Both have in common that they are defined in a machine-readable format. The metamodel of the V-Modell XT for instance is available implemented in XML and as PDE-based domain-specific language [32, 33]. The metamodels can be used to define tool support in manifold fashions (e.g., xSPEM [6], enactment in general [30, 31, 18]).

Flexibility: Flexibility of rich process models is given in several areas. Rich processes’ metamodels (usually in combination with some tools) build process frameworks (also referred as process lines [44, 47, 48]) that allow for customization according to organization- or project-specific requirements. One aspect is the tailoring according to tools. Some process models, such as the V-Modell XT contain only rather generic method content. Concrete methods are implemented in projects using specific tools (e.g., “abstract” quality assurance, which is realized using tools for unit testing). Other process models contain more detailed method content, such as for example the method for object-oriented design and
development in RUP. Appropriate tools support the application of such methods in the project. For instance, (situational) method engineering (SME) [9, 10] covers concepts and languages to construct reusable process assets. However, it remains on a more theoretical level [12]. This is also reflected by the general absence of practice-based research [42]. Nevertheless we can find some concrete approaches, especially using tools [12] for SME, especially to create concrete methods from building blocks.

In terms of flexibility with regards to a metamodel, process customization can be done systematically. Process fragments (i.e. particular methods) can be (1) described in general, and (2) extended according to tools that are used in the projects. Different configurations are possible – one process, which is interpreted and realized using several tools. The introduced PET platform allows to read one process model and exporting it for different outputs (i.e. a management and a development view) represented by different tools.

**Enactment and Business Processes.** While business processes and software development process models differ in some fundamental characteristics, we can find some inspiration in the area of business process modeling (BPM, [52]) that can be transferred to software development models and their enactment. In BPM there are commonly accepted methods and techniques, such as Aris [15] or BPMN [39]. These methods are mature, easy to communicate, and tool-supported. However, BPMN is not (and cannot be) suited to describe projects as it requires a full specification of the process – something that cannot be done sensibly for software development projects.

**Discussion.** Regarding the implementation of software process models in tools numerous contributions can be found. Enactment is covered on several levels, e.g., starting pragmatically on the tools’s sites or formally at the metamodel level of process models. Much work regarding the metamodels targets the improvement of semantics or simulation capabilities. This we this is critical, since i.e. xSPEM pays no attention to the tools that are used in a project environment, but requires those tools to understand the SPEM language. Contributions with respect to tools can roughly categorized into (1) building up “ad-hoc” tool chains of several (smaller) task-specific stand-alone tools, and (2) extending comprehensive ALM tools by some process “execution” capabilities. This we also consider this to be critical as for the first case tools have to be selected and aligned with a process manually, which is in our understanding no enactment. Also the second aspect is critical since most of the ALM platforms only support micro workflows for selected process fragments, e.g., work item tracking, but pay rather no attention to comprehensive process requirements, i.e. the TFS even does not know the concept of milestones by default.

The goal of PET is to close the gap between comprehensive process model (descriptions) and tools commonly used in projects. Therefore, PET makes use of process metamodels and boils them down to an intermediate format. This format is in turn converted to the data models used by the target tools. The selected approach is roughly comparable to business process execution in terms of breaking complex process metamodels down to an easier structure, which can easily transformed to the current data structures relevant tools can work with. In consequence PET does not try to implement some kind of process execution engine, but leaves the “real” enactment to the tools, which themselves provide the execution. PET extracts the configuration information for the tools being used in a project.

2.1. **History and Previously Published Work**

The Process Enactment Tool-Framework basically supports the following two usage scenarios: (1) a single process can be transformed for several tools and (2) multiple processes can be
processed by the framework. PET is based on the results of the project CollabXT [30] and is an integrated framework for flexible coupling of processes and tools. With CollabXT, we explored the options for process enactment. CollabXT’s results were two hard-wired tools to transform the German V-Modell XT [19] to outputs for (1) the Microsoft SharePoint Services [51] and (2) for the Microsoft Visual Studio Team Foundation Server [36]. Limitations of CollabXT became obvious during the further development of the V-Modell XT. A change of the metamodel [49] of the V-Modell XT meant that both tools had to be adjusted accordingly. In consequence, for each new release of the process and also for each release of the addressed target tools, the transformation tools had to be changed. At the same time the existing tools had to be maintained to avoid running into compatibility issues.

From the experiences, a new concept for a more flexible and stable enactment platform was developed [31]. PET now potentially supports all development process models that are based on a formal (in terms of machine-readable) metamodel. Furthermore, it addresses several process-supporting tools and supports one-to-many transformation between the process and tools. Thanks to the decoupling of development process model input processing and tool format output generation, if the metamodel of a development process changes, only the corresponding transformation plugin of PET has to be adjusted. The other modules of the tool can remain untouched.

Previously Published Work. Starting with the project CollabXT we worked on process enactment for a couple of years. The paper at hand presents the current research. The paper summarizes and extends different publications. Noteworthy are the following contributions: In [30] we first introduced the concept of generating tool-based working environments from development process models. [30] summarized the first outcomes of CollabXT. In [28] we extended our work to distributed development projects and discussed the question of how to provide distributed teams with development processes in collaboration tools. Those publications are the basis for the current work, which is presented in the paper at hands. As enactment is just one aspect of support for a project, we introduced PET in [32] and discussed the construction of a seamless tool chain. A seamless tool chain should support the process life cycle and should also respect enactment while process definition. The generation of working environments is a consequence of organizations’ requirements during process customization/improvement projects. In [32] we sketched, how to integrate PET with a process design tool. The paper is focused on PET and does not discuss the integration in process development environments. The PET platform itself is described in detail in [31] from a technical point of view. In the paper at hand we focus to the basic and methodical aspects of PET.

3. Architecture

PET was designed with the intention of providing a flexible and extensible platform that brings together arbitrary processes and tools. We first present the details of this transformation. Afterwards, we describe the concepts necessary to provide this functionality as well as the resulting architecture. Furthermore, we show how a PET plugin is created and give concrete examples for development processes and project-supporting tools addressed by PET.

3.1. Logical Architecture

PET decouples the parsing of the process from the tool-specific translation. To be able to do so we introduce an intermediate model that abstracts from concrete processes and tools. A
transformation from process to tool data now consists of two steps: The input process is first mapped to the intermediate model. In the following only the intermediate data is used to generate the data for the project-supporting tool. This allows us to change the process model parsing logic without touching the tool data generation logic and vice versa. Thus, arbitrary process models can be enacted in arbitrary project-supporting tools as long as the parsing and generation logic uses the intermediate model.

To ensure the extensibility of the platform, both process parsing and tool data generation are handled by separate self-contained modules that are loaded dynamically. Fig. 2 shows the described translation. In PET, the modules to load process models are called process providers and the tool data generating modules are called tool providers.

3.2. Technical Architecture

The logical structure presented before is implemented in a three-part architecture that is shown in Fig. 3. The platform module contains interfaces and functionality needed in all PET components as well as in its plugins. Its most important parts are the intermediate model and the process- and tool provider interfaces. The provider modules are the second part of the architecture. They expose implementations of the platform’s process or tool provider interfaces. The part most visible to the user is the application/wizard as it implements PET’s user interface. Furthermore, the application load the provider modules, allows the user to configure them and controls the steps of the process model to tool data transformation. In the following sections we elaborate on the elements of the architecture.

3.2.1. Intermediate Model

The intermediate model of PET is an abstract representation for a software development process. It is used to decouple PET’s process parsing logic (process providers) from its output generation logic (tool providers). When designing the intermediate model, we had to balance the factors generality and ease of use. Generality describes the expressiveness of the model, while ease of use measures how understandable it is without digging deep into documentation. Obviously, the factors influence each other: an extremely general model allows arbitrary constructs that may be hard to grasp whereas a limited model is comprehensible but can lack expressiveness. When we balanced those two factors during the design of the PET intermediate model, our
main goal was the model’s expressiveness for the tool providers. Our focus was on the output data for the project-supporting tools, not on the input process. We did not intend to create a new generic process metamodel. The only purpose of the intermediate model is to decouple input and output providers. Sect. 6 provides further information on the design of the intermediate model.

The PET intermediate model is divided into two logical parts: (1) the artifact model and (2) the association model (according to [27]). The separation was a design decision based on our experiences in process metamodel development.

Artifact Model. The artifact model is responsible for the definition of process elements. It contains a fixed set of elements of software development processes (Fig. 4) used in practice, which makes it easy to understand. Usually such processes describe who does what, when, how and with which tools.

The artifact model is roughly divided into two parts: (1) the static process elements and (2)
the dynamic process elements. Static process elements represent process elements and structures, which are usually used to build the process’s structure, e.g., roles or work products. Process structures are used to build process documentation and to derive templates, e.g., for project plans or document templates. The second part of the artifact model contains the dynamic elements. Dynamic elements are used to model “active” parts of a process model, i.e. workflows. Process-oriented approaches such as RUP describe concrete steps for the process using workflows. Thus, dynamic elements are used for several processing tasks, but usually skipped for the process documentation, PET considers such elements separately; nevertheless, a connection between the static and dynamic parts of the artifact model is established by the interface IWorkflow in the static part. Table 1 gives a short description for the core elements of the model from Fig. 4. [31] contains detailed information regarding the elements of the artifact model, i.e. concrete properties and structure.

<table>
<thead>
<tr>
<th>Element</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>IProcessElement</td>
<td>base type for all process elements</td>
</tr>
<tr>
<td>IStaticProcessElement</td>
<td>base type for all static process elements (e.g., roles, work products etc. usually contained in the process description)</td>
</tr>
<tr>
<td>IDynamicProcessElement</td>
<td>base type for all dynamic process elements (used for modelling workflows for processes for enactment)</td>
</tr>
<tr>
<td>IArtifact</td>
<td>an artifact of the process model, i.e. the result of an activity</td>
</tr>
<tr>
<td>IDiscipline</td>
<td>a discipline that is used to categorize artifacts, i.e. reports or source code</td>
</tr>
<tr>
<td>ITopic</td>
<td>an element describing the internal structure of (document-based) artifacts</td>
</tr>
<tr>
<td>IActivity</td>
<td>an activity description in the development process</td>
</tr>
<tr>
<td>ITask</td>
<td>a discrete task description that is part of an activity</td>
</tr>
<tr>
<td>IMilestone</td>
<td>a milestone belonging to the development process, i.e. project initialized</td>
</tr>
<tr>
<td>IRole</td>
<td>a role involved in the process</td>
</tr>
</tbody>
</table>

Table 1: Core elements of the artifact model.

The elements included in the artifact model are defined according to our experiences from several process improvement projects and related tool development. Until now the volume proved appropriate; still, extensions are possible by defining new types that are derived from IProcessElement (or a more specialized type).

Association Model. The second part of the intermediate model is the association model that covers the relations between instances of artifact model elements. Associations are modeled in an abstract way, making them general and versatile. Associations may exist between elements of arbitrary types, multiple associations with the same source and destination are valid. Figure 5 shows a simple example for the interplay of artifact model and associations in PET: A given process contains a couple of thousand interconnected elements. PET analyzes the process and

\[1\]Note that in our implementation the model elements are realized as interfaces which is the reason their names begin with the capital letter I. For legibility this first letter is omitted in later paragraphs, so that e.g., for IArtifact we simply write Artifact.
collects the artifacts in a first step. In the second step associations are analyzed. After that PET has knowledge about different dependencies and participating elements. This allows PET for providing specialized views on the interconnected artifact model to address specific needs of the tool providers.

For given process elements $p_1$ and $p_2$, $\text{assoc}(p_1, p_2)$ denotes that there is an association between them. If no additional information is given associations are directed relations. Thus, $\text{assoc}(p_1, p_2)$ means $p_1$ is related to $p_2$. $\text{assoc}$ is not symmetric: $\text{assoc}(p_1, p_2) \Leftrightarrow \text{assoc}(p_2, p_1)$. Further association information is given by the optional attributes $\text{Direction}$ and $\text{Type}$ specifying the association’s symmetry and semantics. They are described in Table 2.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Directed</td>
<td>default behavior, $\text{assoc}_{\text{dir}}(p_1, p_2) \Rightarrow p_1$ is related to $p_2$</td>
</tr>
<tr>
<td></td>
<td>NonDirected</td>
<td>symmetric association, $\text{assoc}<em>{\text{nondir}}(p_1, p_2) \Leftrightarrow \text{assoc}</em>{\text{dir}}(p_1, p_2) \land \text{assoc}_{\text{dir}}(p_2, p_1)$</td>
</tr>
<tr>
<td>Type</td>
<td>Relation</td>
<td>default behavior, general association</td>
</tr>
<tr>
<td></td>
<td>Dependency</td>
<td>$\text{assoc}_{\text{dep}}(p_1, p_2) \Rightarrow p_1$ depends on $p_2$ (e.g. it might be necessary to check $p_1$ after modifications of $p_2$)</td>
</tr>
<tr>
<td></td>
<td>Composition</td>
<td>$\text{assoc}_{\text{com}}(p_1, p_2) \Rightarrow p_1$ comprises $p_2$</td>
</tr>
</tbody>
</table>

Table 2: Association attributes.

Creating basic associations like Task $X$ belongs to Activity $Y$ from scratch for every new process or tool plugin can be tedious. To prevent this unnecessary work PET contains several pre-defined association types that are strongly typed and already have the appropriate attributes set. Some examples are TaskToActivity (Directed, Composition), ArtifactToArtifact (Directed, Dependency) and ActivityToRole (NonDirected, Relation). Just like the elements of the artifact model, this set of pre-defined associations is based on our experiences.
3.2.2. Application and Wizard

As presented in Figure 3, the application, which is also called wizard, is the user interface of PET. It provides the frame for the process and tool providers, controls the transformation and interacts with the users [31].

Application Frame. When the application is started, it searches plugin modules that implement process- or tool provider interfaces and loads them. Furthermore, the application frame provides means to save and load PET transformation project configurations. A transformation project configuration comprises the configurations of all used providers put together and enriched with project-specific meta data. For this purpose PET process- and tool providers contain functionality to serialize and deserialize their configuration. Transformation project configurations save the user the effort of entering the same information multiple times when a process transformation is done more than once. For example the input process may change or another transformation product might be needed. Additionally, the application frame contains a centralized logging system that provides the user with information on the transformations.

User Interface. PET’s user interface resembles a Windows wizard and guides the user through the necessary configuration steps of the transformation. Whenever a process- or tool provider must be configured, its user interface page is loaded from the provider module. More on this is described in the following paragraphs.

Transformation Workflow. We already stated that PET basically creates input data for various tools from existing processes. The sequence of steps necessary to do so is called transformation workflow. Figure 6 shows an simplified version.

The transformation workflow is executed by the application. It defines the order in which the wizard pages are shown and the order in which the provider interface methods are called. We already mentioned that providers have their own configuration pages. When a configure step (see Configure Process Provider and Configure Tool Provider in Fig. 6) is reached, the application starts the respective provider’s individual configuration micro workflow. When this micro workflow has completed, the main transformation workflow is continued. A more detailed view on the transformation workflow is provided in Fig. 7. It is important to note that all specific transformation logic is controlled by the providers. The transformation process itself just controls, which providers can be selected in which stage.

Figure 8 shows an exemplary transformation process. This is a concrete example of the concept shown in Fig. 2. If a new transformation project is created, a process provider needs to be selected. Afterwards, the selected process provider executes its configuration workflow; it could
for example ask the user for the folder paths of necessary input data. Now the process provider’s actual functionality is executed – it parses the input process. Once the parsing is completed, PET moves on to the wizard’s next step in which one or more tool provider(s) must be selected. As for the process provider, the tool providers’ configuration workflows are executed. As the process was already parsed at this point, the tool provider configuration pages can access the process data through the intermediate model. When all providers are properly configured, the actual transformation can be started. While it is running, PET displays status information to the user.

3.2.3. Platform Usage

The previous sections discussed PET’s architecture and showed that most of the transformation logic is contained in the process- and tool providers. In the following we want to present the steps necessary to create a (very basic) example tool provider.
Firstly, we create a new Microsoft.NET library project and add a new public class that implements the interface `IToolProvider` which resides in the platform library `TUM.CollabXT.dll`. The `Initialize` method receives an instance of the process provider which the user selected during the transformation configuration steps. This process provider contains an intermediate model instance which is initialized with the process’ data. For this simple example we only store the process provider instance in a class field called `ProcessProvider`. The `GetConfigPageName` method must return the name of the tool provider’s main configuration page. As we do not want any configuration for the example tool provider, `null` should be returned. The last of the three mandatory methods is called `Process` and should implement the actual generation of output data. To test PET’s functionality we could for example write all artifacts’ names to a file (Listing 1).

Listing 1: Example. How to use PET infrastructure

```csharp
using (var sw = new StreamWriter("artifacts.txt")) {
    ProcessProvider.Artifacts.ForEach(a => sw.WriteLine(a.Name));
}
```

While it may be helpful for debugging purposes to get a dump of all parsed artifacts, real output generators are of course a lot more complex. For a longer description on how to implement process- and tool providers refer to [31].
4. Provider Overview

In the following section we want to introduce some providers that are included in the PET project at Codeplex. PET provides full support for process models based on the V-Modell XT 1.3 metamodel (Sect. 4.1.1) as well as prototypical support for processes based on the SPEM metamodel (Sect. 4.1.2). Furthermore, it generates process templates for the Team Foundation Server, builds site structures for SharePoint, and creates work product document templates in the Office Word format (Sect. 4.2 ff.).

4.1. Process Provider

In the following sections we are introducing the process providers that are included in the PET default package.

4.1.1. V-Modell XT

The first provider we want to introduce is the provider for the German V-Modell XT. It maps a project-specific tailored V-Modell XT instance to the intermediate model (for detailed technical information regarding the concrete mapping, see [31]). The process-provider-specific implementation of the Initialize() methods performs a mapping of the V-Modell XT elements according to Table 3.

<table>
<thead>
<tr>
<th>V-Modell XT</th>
<th>Element description</th>
<th>Mapped to</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Process Module]</td>
<td>Process Modules contain all elements structurally necessary to describe a process. This element is only used indirectly to get list of needed V-Modell XT elements.</td>
<td>–</td>
</tr>
<tr>
<td>[Work Products]</td>
<td>Work Products are used to get the results of the process.</td>
<td>Artifact</td>
</tr>
<tr>
<td>[Topics]</td>
<td>Topics structure and refine work products.</td>
<td>Topic</td>
</tr>
<tr>
<td>[Subtopics]</td>
<td>Subtopics are the structure within a Topic.</td>
<td>Topic</td>
</tr>
<tr>
<td>[Decision Gates]</td>
<td>Decision Gates are the V-Modell XT concept for milestones. They are defined in the project file of the Project Assistant [31].</td>
<td>Milestone</td>
</tr>
<tr>
<td>[Activities]</td>
<td>Activities describe the actions necessary to create the Work Products.</td>
<td>Activity</td>
</tr>
<tr>
<td>[Disciplines]</td>
<td>Disciplines are used to structure Work Products. Logical grouping is used for further work in the tools.</td>
<td>Discipline</td>
</tr>
<tr>
<td>[Roles]</td>
<td>Roles describe who is responsible in the project.</td>
<td>Role</td>
</tr>
<tr>
<td>[Relations]</td>
<td>Relations build the structure of the process by coupling the elementary process elements.</td>
<td>Associations</td>
</tr>
</tbody>
</table>

This provider covers the complete V-Modell XT structure. It was applied in three projects with real world releases of the V-Modell XT, both for the German and for the English derivative of the V-Modell XT.

---

2available at http://pet.codeplex.com
4.1.2. SPEM

The second provider we want to introduce is the process provider for SPEM-based process models. Similar to the provider for the V-Modell XT, the SPEM-provider maps processes to the intermediate model using the mapping shown in Table 4.

<table>
<thead>
<tr>
<th>SPEM Element</th>
<th>Description</th>
<th>Mapped to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Artifacts</td>
<td>Products of work, output of tasks.</td>
<td>Artifact</td>
</tr>
<tr>
<td>Milestones</td>
<td>Describes a significant point in the project.</td>
<td>Milestone</td>
</tr>
<tr>
<td>Activities</td>
<td>Descriptions of work units.</td>
<td>Activity</td>
</tr>
<tr>
<td>Task Definitions</td>
<td>Tasks are descriptions of work.</td>
<td>Task</td>
</tr>
<tr>
<td>Role Definitions</td>
<td>Roles describe who is responsible in the project.</td>
<td>Role</td>
</tr>
<tr>
<td>Domains</td>
<td>Logical grouping of work products.</td>
<td>Discipline</td>
</tr>
<tr>
<td>References</td>
<td>References describe relations between process elements.</td>
<td>Associations</td>
</tr>
</tbody>
</table>

Table 4: SPEM mapping to the intermediate model.

Due to the complexity of SPEM the support is currently “prototypical” and only covers the most relevant process elements.

4.1.3. Intermediate Model Representation

The technical representation of software processes differs dramatically: The V-Model XT for instance is contained in only one XML-file (and a few additional configuration files). A SPEM-based process, i.e. OpenUP is, on the other hand, spread over a number of XMI-files. To provide an integrated approach for process transformation, an abstraction of the technical representation, which is realized by the intermediate model, is necessary. Independent of the original process, the representation in the intermediate model is the same. Figure 9 shows two concrete instances (SPEM and V-Modell XT) of process models as they are interpreted by PET. The figure shows the list of artifacts PET publishes to the tool providers, e.g., the OpenUP instance contains 27 artifact types and 53 activity types. The technical representation also supports the decision to differentiate between process artifacts and associations (Sect. 3.2.1). For instance, for OpenUP PET extracts 4757 different associations.

It’s such representation that is the input for the tool providers. Process elements (the figure shows the artifacts/work products) are parsed and transformed to abstract from process structures. Tool providers use those representation to create the data templates or to call the APIs. Listing 2 gives an example, how the intermediate model is called by a tool provider. At this stage, it is basically irrelevant in which format the original process model was defined – PET completely abstracts from the given representation. In Sect. 4.2.3 the concrete application is shown (two processes are used by one tool provider).

Listing 2: Accessing the PET intermediate model (example)

```java
// _processProvider is given by the intermediate model
IRole projectLeadRole = null;
// Find role project lead
foreach (var role in _processProvider.Roles) {
    if (role.Name == "Project lead") {
        projectLeadRole = role;
        break;
    }
}
```
4.2. Tool Provider

Tool providers are the second key component of PET’s architecture. We briefly introduce the providers in the following sections.

4.2.1. Team Foundation Server

The first provider we want to introduce is the tool provider for the Microsoft Team Foundation Server (TFS). TFS is supported by PET in the version 2005, 2008, and 2010. The tool provider for TFS generates a complete and valid process template in the tool-provider-specific implementation of the \texttt{Process()} method. The provider’s output can be imported into a running TFS instance and serves as template for new projects.
In order to create the TFS process template, the corresponding tool provider implements the micro workflow that is shown in Fig. 10. Basis for the output generation is a TFS meta template which contains the structure of the final template as well as several static files (i.e. images containing the corporate identity of the developer). Furthermore, the tool provider is able to embed a process documentation in the project template. If the documentation shall be included, the folder path for the documentation files has to be specified.

The default TFS work items are not sufficient to represent the information in the intermediate model. PET therefore defines custom work item types. For a detailed description of these types and the rationale behind them refer to [31].

4.2.2. SharePoint

The tool provider for SharePoint generates a team website based on the information provided in the intermediate model. This team website is designed to be a workspace for projects that have to follow the defined process. Generated SharePoint team websites target the management in a project, less the development-focused roles. Fig. 11 shows a website generated by this tool provider. The generated team website contains for example (based on the process model contents and structures):

- A site/sub-site structure according to the logical design of the process model, e.g., a sub-site/sub web per process discipline.

- Work product templates are stored in the sub-webs’ custom lists according to their discipline assignment.

- Activities are instantiated and connected to work products.

In contrast to other tool providers that write their output to the local file system, the SharePoint provider uses the destination platform’s programming interface to directly publish its results on the server.
4.2.3. Office Word

The DOCX tool provider uses generation template documents and the information from the initialized intermediate model to create document templates for artifacts in the Office Open XML Format (OOXML). Generation template documents are *.docx files containing placeholders that determine the structure of the final document template. This allows the user to customize the results generated by the provider. For example it is possible to have the document templates reflect the software development project’s design or a corporate identity.

The configuration page (Fig. 12) gives the user the ability to specify the name and company of the project for which the document templates are generated as well as their author. These information are put into the generated artifacts at the respective placeholders as well as in the documents’ meta data3. Optionally a V-Modell XT template texts XML file can be used to include content in the generated document templates4. Last but not least the artifacts for which document templates are to be generated must be selected in the belonging section of the configuration page.

5. Applications

Currently we have PET applied in two scenarios: (1) application in practice and (2) in education.

---

3In addition to the generation of document templates, an additional software component allows for working with text templates at runtime. A self-contained Codeplex project http://petruntime.codeplex.com provides a Microsoft Word 2007/2010 plugin that makes text templates available. This component requires document templates created by PET due to the contained meta data.

4Note that the use of the V-Modell XT template texts file is completely independent of the used process provider; using it with a SPEM process provider would be possible.
Application in Practice. PET was applied in cooperation projects, i.e. with the German Federal Network Agency. The setting in this scenario included a customized V-Modell XT variant as process model and a Team Foundation server as software infrastructure for development projects. In another pilot application PET was used to generate requirements documents from a formal process description to evaluate the process model and to determine the use of the generated work products/artifacts [35, 34].

Education. PET was subject of two practical courses with students from the University of Applied Sciences Rosenheim. In the first course the students created a process provider that demonstrates the mapping of SPEM-based processes to PET’s intermediate model. The test case to verify the provider’s functionality was to load the OpenUP process [2]. During the second course a new tool provider for SharePoint was created. Both providers have yet to be thoroughly reviewed by our team and then integrated into the default PET package. The process provider for SPEM currently reads SPEM-based processes in order to create document templates, which is applied to OpenUP and Scrum methods (see Sect. 4.1.2).

6. Challenges

This section describes major challenges that we were facing during development of PET and that may be worth sharing.
Intermediate Model Design. A major challenge was the definition of PET’s intermediate model (Sect. 3.2.1). PET is not meant to replace unifying approaches for process metamodels such as SPEM, but to provide a common interface between arbitrary processes and project tools. Therefore, PET’s intermediate model was designed in order to address the tools’ requirements with regards to template functionality or custom data models. The challenge was to break down data models and APIs into a common data structure which can store process contents. Team Foundation Server and SharePoint were the first tools under consideration, providing information regarding artifact structures, e.g., work item types or project structures (see Sec. 4.2.1), and information filtering or user experience (see Sect. 4.2.2).

Mapping an input development process model onto the intermediate model involves a potential loss of information. The intermediate model only includes information that is relevant to target tools. Although PET can provide all considered tools with the required input data, the question if the basic process is “completely implemented” remains unanswered. In terms of the V-Modell XT we can (manually) assess the conformity of the process’s instance. In all the cases up to date we were able to assess conformity based on the original process and its representation.

Migration of Existing Code. When we started the PET project, our first priority was porting the existing functionality of CollabXT (SP/TFS) to the new framework. This involved a great deal of re-engineering. We had to separate the code responsible for parsing the V-Modell XT process model from the parts responsible for writing the SharePoint and Team Foundation Server data.

CollabXT TFS was particularly hard to port into a PET tool provider. The implementation was created by an external developer written in Ruby. This language allows a pattern matching-oriented programming that was heavily used for the processing of the XML structures in the V-Modell XT and the TFS templates. The process transformation logic had to be completely reimplemented.

Another challenge we faced is the integration of code that was created in the course of student papers and theses [32]. While the theoretical results presented are often highly useful, the created source code has to be carefully reviewed. A part of PET that was originally developed in a student project is the Word-DOCX tool provider described in Sect. 4.2.3.

Platform Evolution. As PET is an active research project we faced several challenges that resulted from the platform’s evolution. A lot of work went into the implementation of new features while maintaining stable interfaces. To introduce the new intermediate model in PET 1.1 we had to break backwards compatibility – a layer mapping the old model to the new one would have been very complex, if not impossible for some scenarios. Nevertheless, it was important to us to keep the migration to the new intermediate model as simple as possible for provider developers. In fact it was so simple that we were able to migrate most parts of our process and tool providers automatically (e.g., by refactoring and renaming of used classes). Sect. 7 elaborates on the future evolution of the PET platform.

7. Roadmap

PET reached a stable release status and is continuously maintained. Additionally, some areas for improvement have been identified for further development. Besides minor changes and improvements, e.g., an implementation of a tailoring algorithm for the V-Modell XT, PET will be extended with additional features and components.
From the practical courses a first version of a process provider for SPEM is available. One of the next steps will be to finish the quality assurance of that provider and integrate it into the standard providers delivered with PET.

PET’s current architecture assumes the existence of exactly one process- and one or more independent tool provider per transformation. An idea we have yet to realize is the chaining of multiple tool providers, i.e. using the previously created output as input data. A scenario for this is for example the generation of work product templates which are then uploaded to a SharePoint portal that will be used as the project workspace.

With respect to [32] we plan to integrate PET with PDE to provide a seamless tool chain to support the whole process life cycle. PDE acts as design tool for process models. PET will be integrated into PDE as plugin, enabling the user to export the process model directly from the design tool into a tool infrastructure.

8. Conclusion

Based on first experimental tools developed in the CollabXT project, we have developed PET to transform an input software development process model into data formats usable by tools for project execution. A flexible and scalable architecture of PET now allows to create process enactment solutions for potentially any metamodel-based software development process and for any tool that has a publicly available data format. We now have a mature implementation for the V-Modell XT and a prototypical one for SPEM-based development processes as well as a number of output transformation plugins. We have shown the applicability in a number of projects with government and industry partners. The modular architecture enables us and other contributors to extend the capabilities of PET and the number of interfaces for both input processes and for output tools.

References


[38] Rombach, D., 2005. Integrated Software Process and Product Lines. In: Li, M., Boehm, B., Osterweil, L. J. (Eds.),


