Flexible Process-Tool-Integration

Marco Kuhrmann, Georg Kalus, Manuel Then

TUM-I1005
Dezember 10

Druck: Institut für Informatik der
Technischen Universität München

Flexible Process-Tool-Integration

The *Process Enactment Tool Framework*

Marco Kuhrmann, Georg Kalus, Manuel Then
Technische Universität München
Institut für Informatik, Software & Systems Engineering
Boltzmannstr. 3
85748 Garching, Germany
kuhrmann@in.tum.de, kalus@in.tum.de, then@in.tum.de

Summary

Process enactment is a challenging task. Projects differ in volume, goals and techniques. There is no common methodology to "implement" a development process in a tool or a set of tools. The Process Enactment Tool Framework (PET) is a tool set that supports the transformation of a given formal development process into a format that project tools can work with. In this report we introduce PET, describe the basic architecture and give short tutorials how to work with the platform.

Keywords

Sofware Engineering, Software Development Process, V-Modell XT, Microsoft Office, Microsoft Sharepoint, Microsoft Team Foundation Server

CR-Classification: D.2

Contents

List of Figures

List of Tables

1. Introduction

Using tools for supporting the the implementation of (development) process models[1] can have influence on the acceptance and the applicability of the process. Tools can also be useful to control and enforce the application of the process regarding its regulations. But, the provision of adequate tool support is a challenge. In [KK08b] a first solution was proposed, which uses automatic transformation of a given process model for a particular tool. The advantages were already discussed in [KK08a, Kuh08c, KKD08, Kuh08b]. However, only the first step is done. Further points have to be discovered, such as:

- evolution of tools,
- evolution of the process models, and
- variability in combining tools and processes (including several versions).

The *Process Enactment Tool-Framework*[2] (PET), which is presented in this report, is based on the results of the project *CollabXT* [KK08b]. PET is an integrated framework for flexible coupling of processes and tools. It is no longer focused on the German V-Modell XT [FHKS09] and some selected tools. PET targets all process models that are based on a formal meta model. Furthermore, it addresses several process-supporting tools and supports one-to-many transformation between the process and tools.

1.1. Process Enactment

We define *process enactment* as the execution of processes or process elements by tools. In the context of a software development process, enactment means that tool support for management and development tasks according to the process description and structure is provided.

To enact a process, appropriate process elements (artifacts) have to be identified. Usually it is not helpful to give hundreds of pages process documentation to the team. Project teams rather need selective support for relevant tasks and artifacts. Process enactment also does not mean simply to provide electronic process documentation. A proper support is often independent from documentation. Structures and artifacts are more important as they define data models, which can be adopted by tools.

Aspects of Enactment

Many aspects have to be considered if a process enacted is intended. We give a selection of the most important ones. Further aspects do exist, but they depend on concrete requirements and on capabilities of the processes and tools.

Tools: Existing tools usually provide features that can be used for enactment. Tool-specific features have to be analyzed and should used when suitable. Process structures have to be transformed if necessary. Enactment should not touch existing and habitual working environments (*Zero-impact* strategy). This would have impact on the acceptance.

Documentation: Process documentation should be limited to a minimum of necessarily required content. Unnecessary documentation has to be removed. Furthermore, a re-structuring of the documentation may be required, if other than standard access is required. Not all elements comprised in processes are necessary for a particular setting. A tailoring has to filter the process to remove process parts not utilized.

Views: If possible, multiple views should be provided to respect the different roles participating in a project. This means that one process should be enacted in different tools; e.g. specialized tools may be used for programming and project management (*Single-source* concept).

It has already been mentionen that first steps were done in the project CollabXT, which specifically targets the German V-Modell XT and the tools *Microsoft SharePoint* (SP) and *Visual Studio Team Foundation Server* (TFS) [Mic07]. CollabXT examined

1 In this report we use only the term *process* to denote (software) development processes.
2 Online: `http://pet.codeplex.com`

- what options for process enactment are available, and
- what further requirements have to be fulfilled from projects' point of view.

Templates vs. Runtime-support

An important result of CollabXT was the identification of a conflict: As we stated above, touching the target environment should be avoided. In consequence this requires the target tools to be *process aware*. In that case, templates can be generated from a given process, which contain all information to configure the target tool. However, templates often only contain structure definitions and declarations of initial process elements; the possibilities for support are determined by the target tool. To give an example: TFS only supports work items but contains no logic to create or handle complex work item structures as possibly described by the process to be enacted. So following the *Zero-impact* strategy, the possibilities of process enactment are limited.

Even in case that customizing the target tools is not permitted, sophisticated runtime support can still be provided. Referring the previous example, additional wizards can be created for the handling of complex structures. In fact tool customization is not advised as it may have impact on the enactment's acceptance and will result in increased IT efforts (delivery of patches, add-ons etc.).

PET is primarily designed for the *Zero-impact* strategy and creates templates for process-aware tools. Currently no runtime-support is provided.

1.2. Objectives of PET

The goals of PET are to

- connect arbitrary processes and tools, to
- minimize dependencies between concrete processes and tools, and to
- provide an open and flexible framework for process enactment.

The framework repeals the necessity of defining use cases as done by CollabXT. Processes and tools are abstracted so that they can be coupled in any order. Furthermore, other processes than the V-Modell XT should also be supported out of the box.

1.3. Contribution

This report introduces the Process Enactment Tool (PET). It is structured as follows: Chapter 2 describes the architecture of the PET framework in detail. As an integral part of this architecture, the intermediate model that abstracts processes and tools is presented. In chapter 3 the reference implementation of PET is discussed. In doing so we describe the PET's user interface as well as the process- and tool providers for the V-Modell XT, SP, TFS and Microsoft Word. The appendixes A and B introduce the usage of the PET framework and describe the process- and tool provider implementation process.

2. Process Enactment Tool Framework

The Process Enactment Tool Framework (PET) is a modular process transformation platform. It is able to analyze and filter information given by a formal development process and transform it according to the needs of various target tools. In principle PET is able to work with any meta model-based process.

PET's core comprises an abstract intermediate model for processes and an application framework that controls the transformation. The *intermediate model* is used as an abstraction between the input process, which is mapped to it, and target tools, that use the model. Using the intermediate model, process and tools are decoupled so that combining arbitrary processes and tools becomes possible. Sect. 2.3 elaborates on the intermediate model (also called *core* model).

In PET the logic for the import of processes as well as the export of tool input data is realized as plugins. The first plugin type is called *Process Provider* (Sect. 2.4). Process providers read, filter and import concrete process models into the format of the intermediate model. The second plugin type are *Tool Providers* (Sect. 2.5). Tool providers transform data from the intermediate model to the tool-specific format. The *application framework*, which is described in Sect. 2.2, manages the whole process' transformation; it controls the plugin execution and provides access to the intermediate model.

Overview

2.1. Concept and Architecture

The architecture of PET is based on the experiences from the project CollabXT. CollabXT is the name of two self-contained tools to integrate the V-Modell XT with Microsoft SharePoint (SP) and Team Foundation Server (TFS). Introducing the release 1.3 of the V-Modell XT [FHKS09] the transformation algorithms would not have worked any longer. Among other reasons, the estimated effort for maintenance led to a complete re-design – PET.

PET Components and Plugins

Figure 2.1 shows the architecture of PET. PET is designed around a core that contains all common functionality which is required for process data exchange. This includes the interfaces of the intermediate model (see Sect. 2.3) as well as functionality that supports its easy use. Process-specific functionality is put into self-contained components that are loaded dynamically depending on the context. *Process providers* – on the left in the figure – are components (realized as plugins) that connect processes to the PET meta model. The process provider's task is to read, filter, and to import the process into the intermediate format of the core model, thus it abstracts from a concrete process on the basis of the underlying meta model. Simply spoken, a process provider is a sophisticated import filter. On the other side of the transformation chain are the *Tool provider* components. A tool provider abstracts from a concrete tool. Its task is to take the data from the intermediate model and transform it into a format, the target tool can handle. The management of this transformation process as well as all user interaction are handled by the application frame, which Sect. 2.2 elaborates on.

Hint: This architecture solves the issue that process or tool version changes could impact the transformation process. If there is a new release of either the process or the tool is available, only a single new or updated plugin has to be provided. In fact several plugins for processes or tools can be provided to address specific requirements, such as realizing specialized views.

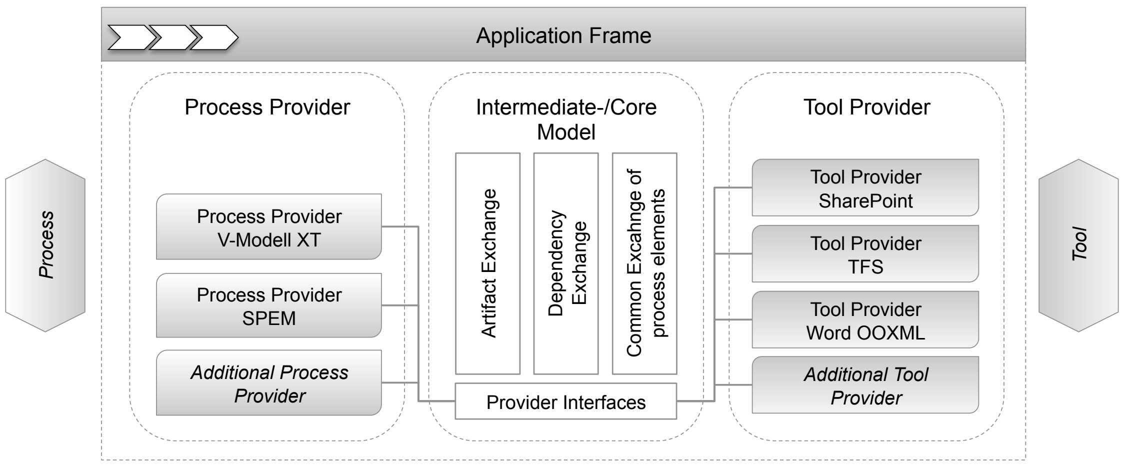


Figure 2.1.: Architecture of the Process Enactment Tool Framework.

2.2. The Application Frame

PET's whole concept is build around the intention to connect processes and tools on a generic basis. As their representations – the process- and the tool providers – do not know of each other's existence, a component that joins and manages them is needed. This is what PET's application frame does.

This section describes the application frame's architecture, its underlying workflow and some user interface considerations.

2.2.1. Functionality

It has been mentioned before, that the application frame manages the process transformation workflow and the providers' life cycles. In detail it provides the following functionality (ordered by their occurence in the transformation workflow):

Load process- and tool providers. The application frame must scan the plugin directories for valid process- and tool providers and load them.

Graphical user interface. To allow an easy usage, PET provides a GUI that allows the selection and configuration of the providers in the transformation process. Furthermore, PET displays progress status information during the transformation. See Sect. 2.2.2.

Infrastructure for loading and storing a transformation project's configuration. Often a process transformation is necessary more than once. E.g. the input process may change or another transformation product is needed. To assist the user in this case, PET's process- and tool providers have the functionality to serialize and deserialize their configuration (see Sect. 2.2.4 and Sect. 2.4). However, the application frame must invoke and manage this functionality.

Status-, warning and error logging. In our examination of various process enactment software on the market we found that error handling and especially meaningful logging is often a problem. A goal for all process- and tool providers is to give helpful information in case of errors. To make this information accessible for the user, the application frame contains status-, warning and error displaying and logging functionality.

Fig. 2.2 illustrates, how the application frame joins all the logical components of PET.

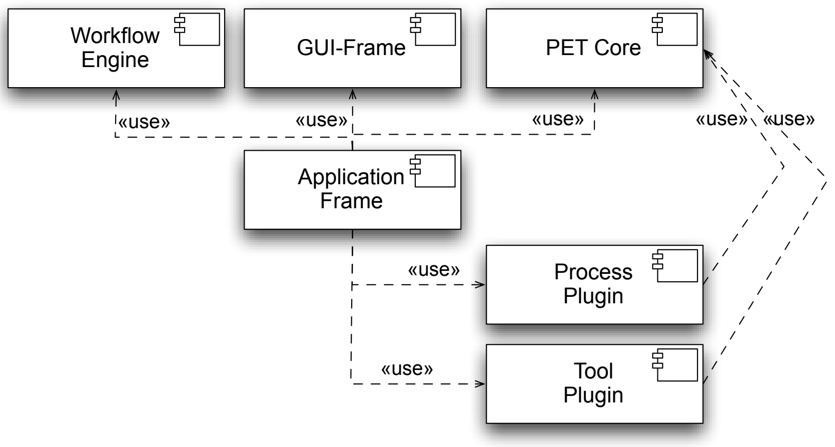


Figure 2.2.: Components of Process Enactment Tool Framework

2.2.2. User Interface

The user interface is given by an application that is close to the well-known wizard dialogs. The GUI-Frame only provides a few fixed dialogs, mainly responsible for controlling the services such as loading and storing project data. Furthermore, navigation through the steps of the wizard is provided; it is realized by means of "Next" and "Previous" buttons. The state and activation of the buttons can be controlled by the plugins as described in App. B.3.

2.2.3. Transformation Workflow

The user interface provides a simple, linear flow. The user is assisted during the particular steps. While the application frame contains some fixed steps (provider selection etc.), most steps that are available are configured by the plugins, which the user selects during the configuration workflow. The whole transformation workflow (that includes the configuration workflow) is shown in Fig. 2.3.

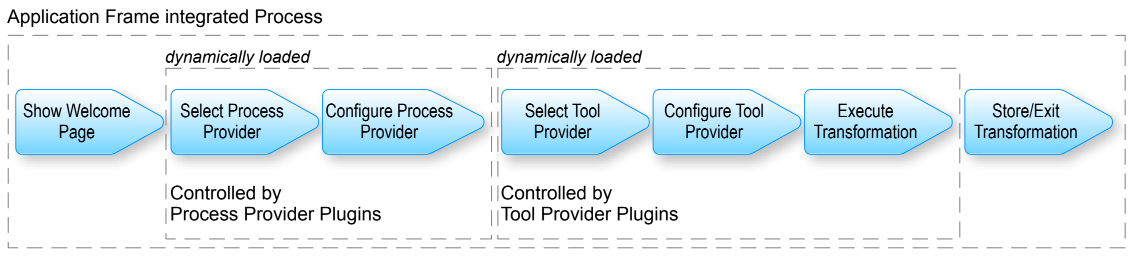

Application Frame integrated Process
dynamically loaded
dynamically loaded
Show Welcome Page
Select Process Provider
Configure Process Provider
Select Tool Provider
Configure Tool Provider
Execute Transformation
Store/Exit Transformation
Controlled by Process Provider Plugins
Controlled by Tool Provider Plugins

Figure 2.3.: Abstract transformation workflow of the Process Enactment Tool Frameworkwizard

A more detailed view is given in Fig. 2.4. It provides a view "inside" the workflow that is run for process and tool providers. Furthermore the workflow gives an impression which operations are executed by the framework. Note that the plugins' exact configuration pages are abstracted here.

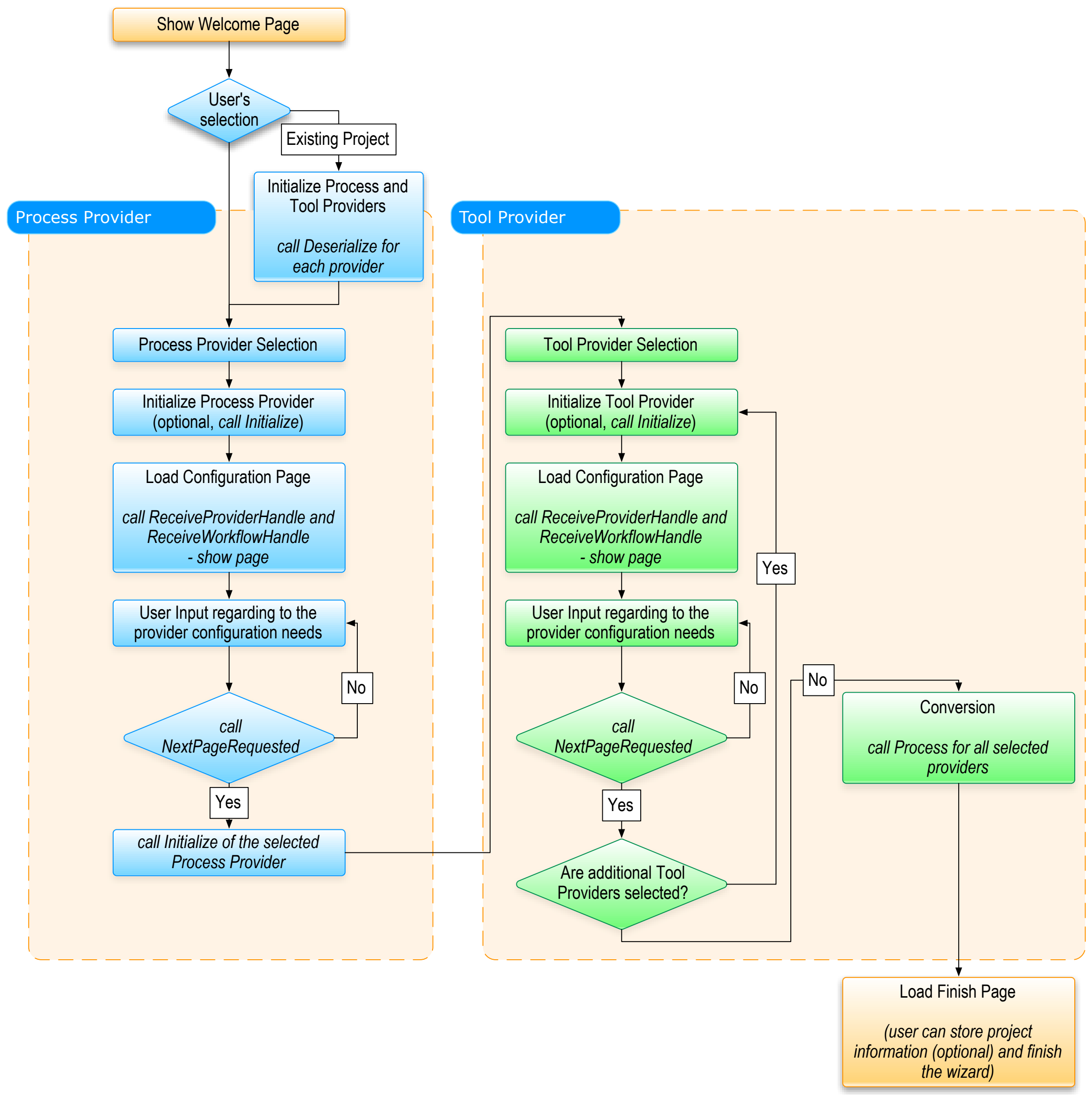

Show Welcome Page
User's selection
Existing Project
Initialize Process and Tool Providers
call Deserialize for each provider
Process Provider
Tool Provider
Process Provider Selection
Initialize Process Provider (optional, call Initialize)
Load Configuration Page
call ReceiveProviderHandle and ReceiveWorkflowHandle - show page
User Input regarding to the provider configuration needs
No
call NextPageRequested
Yes
call Initialize of the selected Process Provider
Tool Provider Selection
Initialize Tool Provider (optional, call Initialize)
Load Configuration Page
call ReceiveProviderHandle and ReceiveWorkflowHandle - show page
User Input regarding to the provider configuration needs
Yes
No
No
call NextPageRequested
Yes
Are additional Tool Providers selected?
Conversion
call Process for all selected providers
Load Finish Page
(user can store project information (optional) and finish the wizard)

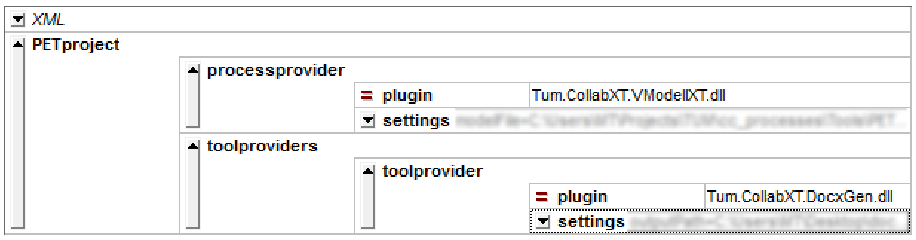
Figure 2.4.: Internal transformation workflow of Process Enactment Tool Framework.

Each plugin can provide several configuration pages and also several self-contained processing stages. The next step in the "global" workflow is automatically performed if there are no further "internal" steps of a plugin are available. The management of the workflow and its representation at the user interface is done by the class `ConversionWorkflow` . This class implements a simple but easily extensible workflow algorithm. Detailed information about PET's whole workflow is given in Fig. 2.4.

2.2.4. Project File Format

PET uses the widespread XML format for its project files. This allows for very good support in most programming frameworks and for human readability.

Fig. 2.5 shows the schema of a sample project file. All PET project files contain the root node `PETproject`, which has two children – the nodes `processprovider` and `toolproviders`. The latter contains a `toolprovider` child node for each tool provider that has been selected in the transformation project.



```
XML
PETproject
    processprovider
        plugin      Tum.CollabXT.VModellXT.dll
        settings    [illegible]
    toolproviders
        toolprovider
            plugin      Tum.CollabXT.DocxGen.dll
            settings    [illegible]
```

Figure 2.5.: Schema of PET project files.

Both the `processprovider` and the `toolprovider` nodes follow the same simple schema: they contain the attribute `plugin` that is set to the assembly name of the respective provider and have one child node `settings`. The node `settings` is written and read by the `Serialize` and `Deserialize` methods (see Sect. 2.4) of the respective provider; PET imposes no schema requirements for this node.

2.3. The Intermediate Model

As described in Sect. 2.1 all imports and exports are abstracted by PET's *intermediate model* (also called core model). The intermediate model is an abstraction of common elements used in process models to realize the flexible combination of arbitrary process- and tool provider plugins. Process provider plugins use the core model as target and store the process data into it. This process information is then provided to the tool providers. They use the core model structures to build the target formats for the tools under consideration.

The intermediate model (shown in Fig. 2.6) is compact and consists of two logical parts:

- the *artifact model*, which contains type definitions for the process elements of interest, and
- the *association model*, that contains type definitions for associations between artifacts.

The basic idea behind this is to separate process content and process structure to gain a common model, which can be used to store data from different inputs. Artifacts and associations are separate model elements. This is necessary for building flexible processes [Kuh08a].

In the following we describe the components provided by the core model in detail. An overview of the artifact model is given in Tab. 2.1. The association model is described in the following.

Basically we understand *all* process elements as process artifacts. *Process artifacts* can occur in different shapes, i.e. work products, activities and so on. Common process artifacts such as roles are already defined in the core model. The types `IProcessElement`, `IStaticProcessElement` and `IDynamicProcessElement` are meta types only used in the type hierarchy. Tab. 2.2 lists their properties.

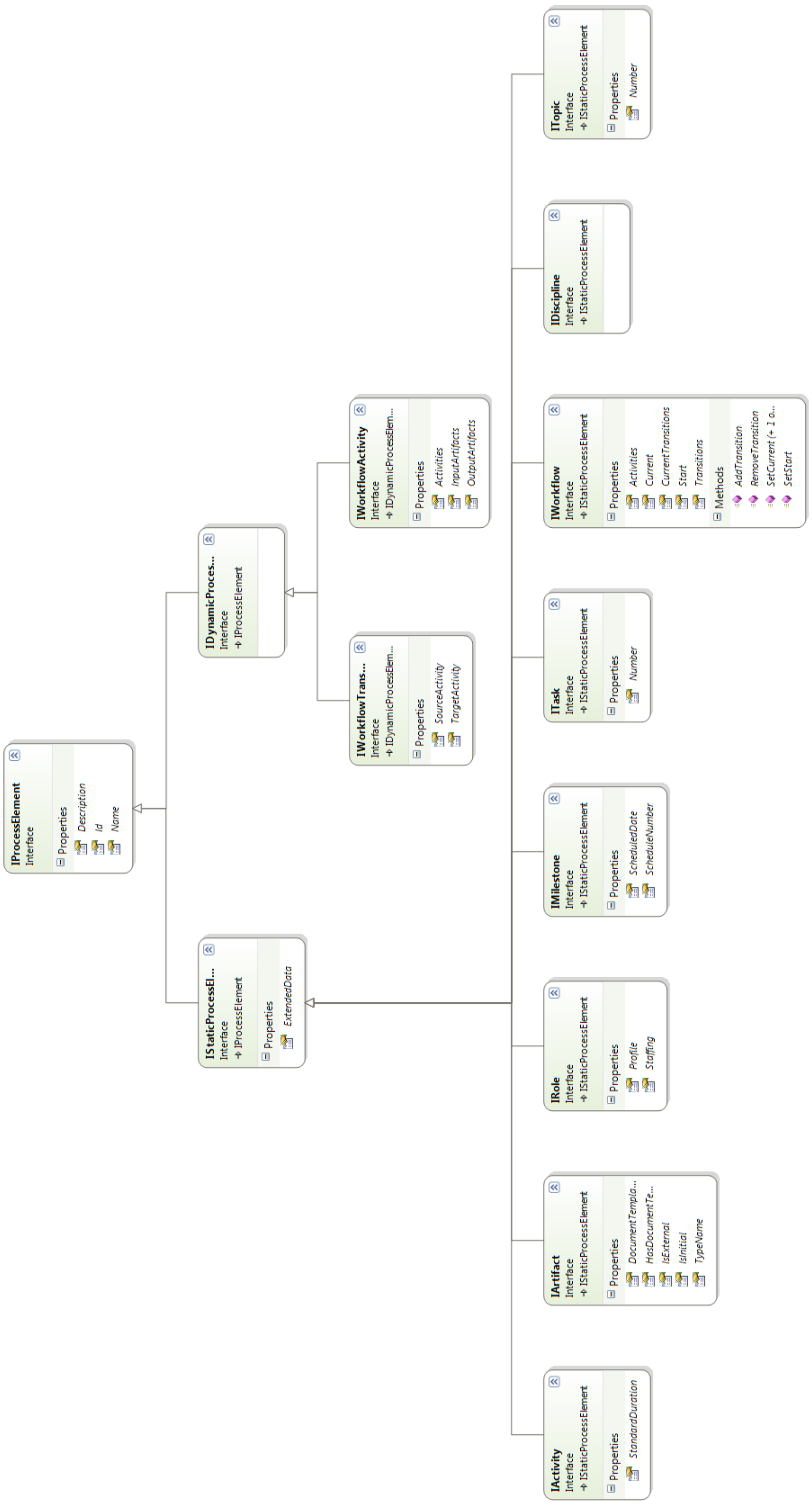


Figure 2.6.: The intermediate model of PET

Element	Description
IProcessElement	base type for all process elements
IStaticProcessElement	base type for all static process elements (e.g. roles, work products etc. usually contained in the process description)
IDynamicProcessElement	base type for all dynamic process elements (used for modelling workflows for processes for enactment)
IActivity	a described activity in the process model
IArtifact	an artifact of the process model, e.g. a work product
IRole	a role
IDiscipline	a discipline used for structuring artifacts
IMilestone	a milestone
ITask	a discrete task as part of an activity
ITopic	a structuring element for (document-based) artifacts
IWorkflow	the container for modelling workflows
IWorkflowActivity	an activity of a workflow
IWorkflowTransition	a transition between workflow activities
IView	views are collections of process elements (used for tailoring)

Table 2.1.: Core components of the intermediate model

Property	Description
Id	unique identifier of the element
Name	display name of the element
Description	textual element description
ExtendedData	(IStaticProcessElement only) key-value collection of properties unique to a certain process (and thus not part of the intermediate model)

Table 2.2.: Properties of IProcessElement, IStaticProcessElement and IDynamicProcessElement

2.3.1. Artifacts

Artifacts [1] are used to model process elements such as work products, which for us is the primary use case. In consequence, work products, outcomes or deliverables (see RUP [KK03]) are mapped to this type of the core model. Tab. 2.3 lists the basic properties of the interface `IArtifact`.

Property	Description
TypeName	the artifact's type
IsInitial	indicates, if the artifact already exists at project start
IsExternal	indicates, if the artifact is created outside the project
DocumentTemplate	path of the template document for this artifact
HasDocumentTemplate	indicates, whether a template document exists

Table 2.3.: The interface IArtifact

2.3.2. Topics

Topics are elements used to structure complex document-based artifacts. This concept was introduced in PET as an equivalent to the respective concept of the V-Modell XT. The intention is to structure documents, but topics can also be used to model complex artifact collections such as deliveries (e.g. consider an artifact *Delivery*, whose topics abstract the comprised parts). Note the difference between topics and sub-artifacts: topics are a structural element of artifacts while sub-artifact associations (see Sect. 2.3.9) model separate artifacts that may exist independently.

1 Artifacts in this context are not to be confused with process artifacts as described previously.

Property	Description
Number	topic order in case an artifact contains multiple topics

Table 2.4.: The interface ITopic

2.3.3. Disciplines

Disciples are logical, content-related containers that comprise artifacts (e.g. work products and associated process elements). This concept can be found in several process models such as the V-Modell XT or the RUP. As depicted in Fig. 2.6 `IDiscipline` does not contain additional properties.

In principle a discipline is like a flag that can be assigned to other process elements for grouping. PET provides an alternative concept called *View* that also acts as grouping mechanism but is free in its application. Views can group elements not only for logical reasons but also for combining elements e.g. for tailoring. That way the PET framework can provide specific tailoring options beyond the original process' capabilities.

2.3.4. Milestones

In PET milestones are used in the sense of project management [Bur02]. They are a means of planning and controlling in a project. Consequently, they contain scheduling-relevant information, such as dates. If relevant for the output data, they can be used to gain first information for the derivation of a project plan. See Tab. 2.5 for the properties of `IMilestone`.

Example: The V-Modell XT contains the tool *Project Assistant*, which initializes projects and allows first planning activities to derive an initial project plan. This information can be processed by PET.

Property	Description
ScheduledDate	the scheduled (finishing) date of the milestone
ScheduledNumber	the number of the milestone to determine an order (comparable to work break down structure numbers as known from Microsoft Project)

Table 2.5.: This interface IMilestone

2.3.5. Activities

An activity is a container that groups several tasks (Sect. 2.3.6). In PET activities are designed similar to the respective concept of the V-Modell XT. Each activity describes the finalization of a work product. So an activity is used as an element for the planning of a work product's creation[2]. Tab. 2.6 explains the property introduced by `IActivity`.

Property	Description
StandardDuration	initially scheduled duration for the activity

Table 2.6.: The interface IActivity

2.3.6. Tasks

Activities in the context of PET are coarse grained and basically intended to act as an means of controlling. Concrete tasks (e.g. necessary for work product creation) are not targeted. To model detailed information of an artifact's creation and portions of work, *tasks* are used. Tasks are atomic portions of work. They are assigned to an activity and are executed in its context (e.g. collecting functional requirements happens during requirements elicitation). See Tab. 2.7 for a description of the property introduced by `ITask`.

2 The V-Modell XT states a 1...0/1 association between work products and activities. PET is not limited to this restriction so activity-driven processes such as RUP can also be mapped.

Property	Description
Number	means of ordering the tasks within an activity

Table 2.7.: The interface ITask

2.3.7. Roles

Roles are used to map the respective concept of the considered process models. Roles contain descriptions for abilities or staffing requirements which people, who work in a project, must fulfill. For a list of the properties in `IRole` refer to Tab. 2.8.

Property	Description
Profile	description of the tasks a role has in a project
Staffing	requirements to people, who are in that role

Table 2.8.: The interface IRole

Hint: Structures of organizations or teams are not covered by PET's role concept. In the context of tools PET's roles are usually mapped to user groups (e.g. for security policies). To give an example: role descriptions of the V-Modell XT can be mapped to user groups of TFS.

2.3.8. Workflows

PET's intermediate model contains a structure model (including several process elements as described before) and a dynamic model. Considering activity-driven process models such as RUP, concrete activities are described using workflows. PET contains a simple set of types in the core model (Fig. 2.7) covering workflow modeling needs, e.g. the workflows themselves (located in the static model to include workflows for certain artifacts), workflow activities and transitions.

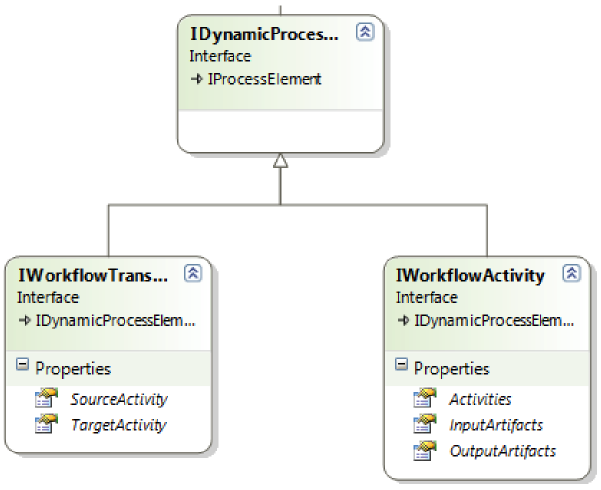


Figure 2.7.: The sub-model for workflows in PET

Hint: It is important to know that this feature is new in the meta-model. The available process and tool providers make no use of this feature at the moment.

2.3.9. Associations

In PET process artifacts and associations between them are defined separately. Thus, a process model can define a set of process artifacts independently of their structures and associations.

PET realizes associations by defining the generic class `Association` that implements the interface `IAssociation` (Tab. 2.9). Based on this class, content-related as well as structural associations can

Property	Description
Id	unique identifier of the association
Name	association name
Description	textual description of the association
ExtendedData	key-value collection of properties unique to a certain process
SourceId	identifier of the source element
SourceType	type of the association's source element
DestinationId	identifier of the destination element
DestinationType	type of the association's destination element

Table 2.9.: The interface IAssociation

be created. The current release of PET pre-defines a set of associations that can be found in almost all process models (see Fig. 2.9). Each association (instance) can hold additional data, e.g. a description or an extended data set, which contains native process data.

The basic idea of PET associations is to provide a flexible and customizable approach to relate process artifacts [Kuh08a]. For given process elements p_1 and p_2 the relation *association* (p_1, p_2) denotes that there is an association between those two process elements. We distinguish two types of associations: content-related and structural ones (refer Fig. 2.8).

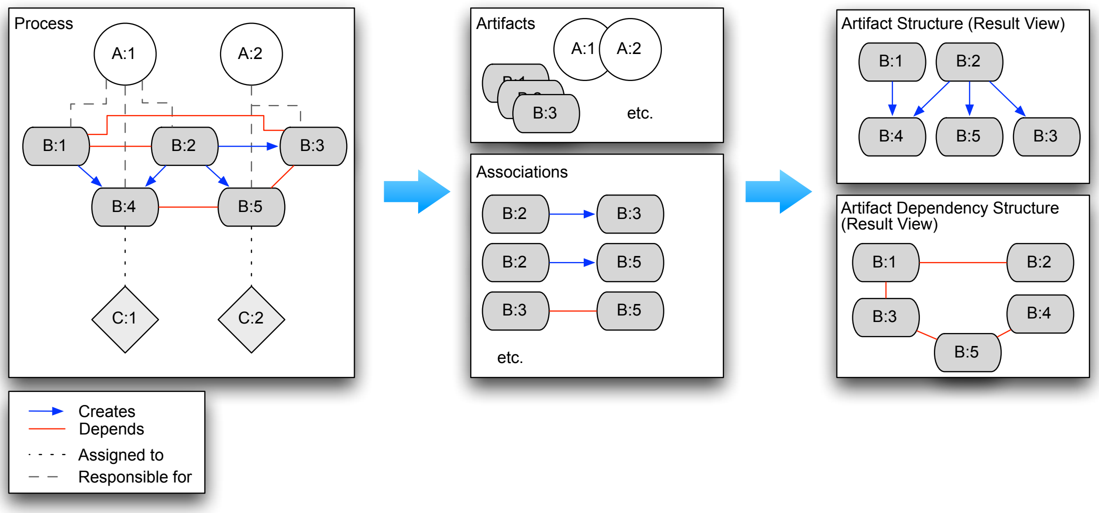


Figure 2.8.: Concept of Associations in PET

Each process element is of a type $t \in Type$, where $Type$ is the set of all Types of process elements of the process model. For content-related associations we define that $typeof(p_1) = typeof(p_2)$. An example for a content-related association is a generative dependency between two work products. Fig. 2.8 shows how PET understands associations (here content-related): A given process may contain a couple of thousand interconnected elements. PET analyzes the process and 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, e.g. the creational structure or the "raw" content dependencies.

If $typeof(p_1) \neq typeof(p_2)$ we call the association structural. Structural associations are used to construct a process model, e.g. to assign work products to roles or to activities. Fig. 2.8 shows this, but not the detailed processing steps.

In the following paragraphs we describe the pre-defined associations in detail. We group the description according to the classification of content-related and structural associations.

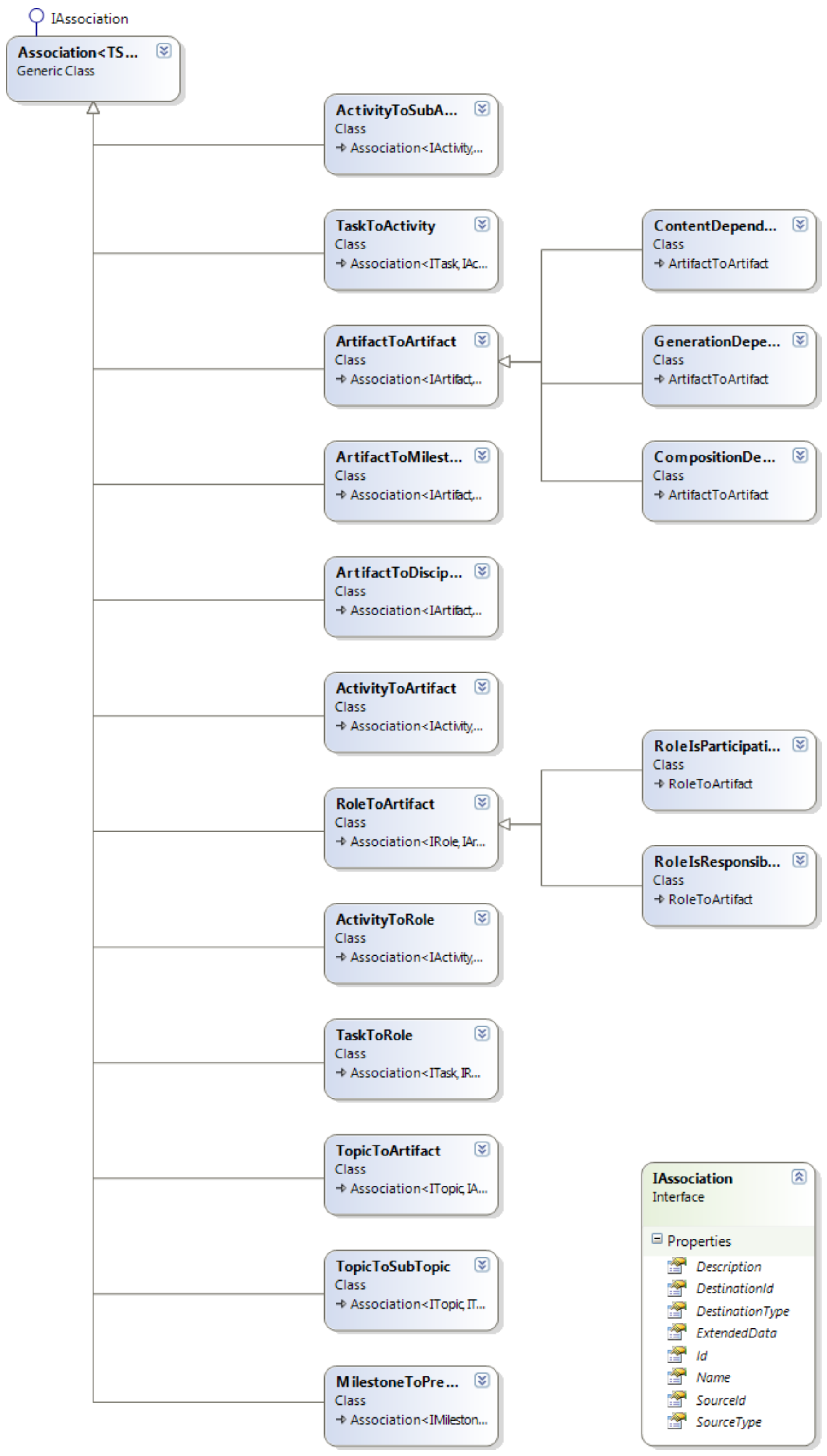


Figure 2.9.: Associations of the PET framework

Association Meta Data

Each association type may hold a set of meta data, indicating its type and direction. The meta data is realized by the attribute class `AssociationKindAttribute` (Fig. 2.10). The enumeration `AssociationDirection` states, whether the association is *directed*, *non-directed* or something else.

The enumeration `AssociationType` describes the semantics of a particular association type. A simple *relation* only relates two process elements p_1 and p_2, regardless of the direction. A *dependency* relates two process elements that are usually of the same type. Furthermore, a dependency is usually directed: $generative\,(p_1, p_2)$ with $generative \in Associations$ expresses that an instance of p_1 generates instances of p_2 – not the other way around. The last type of association we cover is the *composition*: $composition\,(p_1, p_2)\,, composition\,(p_1, p_3)$ with $compostion \in Associations$ express that p_1 comprises the process elements p_2 and p_3.

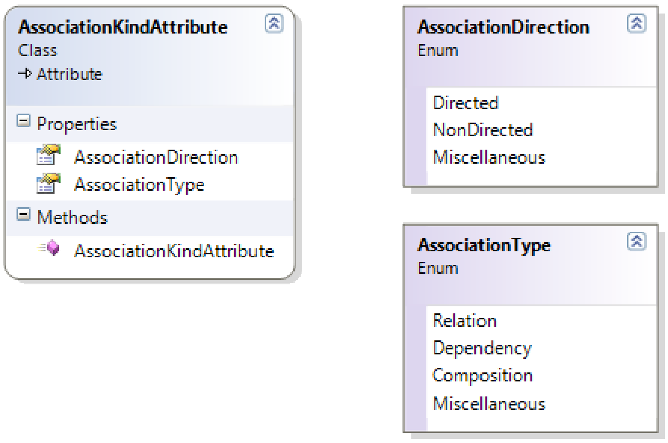


Figure 2.10.: Types for association meta data

The usage of the `AssociationKindAttribute` is optional. However, all pre-defined association types of PET have it. Furthermore its usage is highly recommended for any extension of PET with new association types as it allows the framework and the tool providers to understand the new association's semantics.

Content-related Associations

Content-related associations in PET usually inherit the class `ArtifactToArtifact` (Fig. 2.9). By default three types are pre-defined (see Tab. 2.10). As stated above, content-related dependencies are used to model associations between process elements of the same type, e.g. work products.

Example: An example for such association types is given by the *dependencies* of the V-Modell XT. They express dependencies between the work products involved in such an association. If a work product regulates the condition of the creation of other work products, a *generative* work product dependency is instantiated. Another example is given by the compositional structure of the system under construction: If a system consists of a set of subsystems and/or components, a structural dependency between the particular system's parts is instantiated.

Association type	Description
ContentDependency	If the content of artifact p_1 is changed, p_2 must be updated.
GenerationDependency	The creation of artifact p_1 means that p_2 must be created.
CompositionDependency	Artifact p_2 is a part of p_1.
ActivityToSubActivity	Relates activities in a hierarchical oder to create complex activity structures (specialized composition only for activity containers).

Table 2.10.: Content-related associations in PET

Hint: Due to the flexibility of the association framework it is not mandatory to inherit the class *ArtifactToArtifact* to create new content-related dependencies. The association type *SubActivityToActivity* is an example for that. Principally this association could be seen as a structural one. However, as it creates a hierarchy of activity containers it is indeed a content-related one. So with PET it is possible to create common and "special" associations – the core model is no corset.

Structural Associations

Each process element addressed by the framework implements the interface `IProcessElement`. So for each element an unique identifier, a name, an extended data field, and a description is available. Associations between process elements are established using association classes (see Fig. 2.9). As mentioned above, structural associations relate process elements of different types (e.g. work products and activities).

By default, PET provides a set of pre-defined structural associations (Tab. 2.11), covering standard use cases for constructing process structures. In case of processes that require additional association types, extensions using the base class `Association` can easily be defined.

Association type	Description
TaskToActivity	Relates a task to an activity (composition, an activity is a container for fine grained tasks).
ArtifactToMilestone	Assigns an artifact to a milestone.
ArtifactToDiscipline	Relates an artifact and a discipline to provide for grouping.
ActivityToArtifact	Relates an activity and an artifact to define, what activity has to be executed to create a particular artifact.
RoleIsParticipatingInArtifact	Associates a role and an artifact to express that a role participates in an artifact's creation process.
RoleIsResponsibleForArtifact	Relates an artifact and a role to model which role is responsible for the artifact's creation process.
ActivityToRole	Relates an activity and a role to model, who performs a particular activity.
TaskToRole	Relates a task and a role to model, who performs a task.
TopicToArtifact	Relates topics and artifacts to model, how a particular artifact is structured (usually used for modeling complex artifact structures, such as documents).
TopicToSubTopic	Relates topics in a hierarchical order to create complex artifact structures.

Table 2.11.: Structural associations in PET

Mapping of Arbitrary Process Models

PET is designed to cover different process models. A formal prove is out of the scope of this report. However, considering process models for Software development we can argue as follows: Software development processes only contain a small set of process artifacts to be considered. In principle we have to consider:

- (complex) activities,
- (fine grained) tasks,
- work products (including results, outcomes, deliverables), and
- resources.

Furthermore we have to take into account the relations between the those process artifacts. Looking at common development processes, all necessary process elements can be mapped to the PET intermediate model. Furthermore its flexibility enables the definition of additional process elements as well as associations (see Fig. 2.6).

2.4. Process Provider

Process providers are the interface between the (input) process model and the intermediate model. A process provider's task is to map a given input process model to the intermediate model described before (Sect. 2.3) as efficiently as possible. In this section we describe the process provider concept in detail. Fig. 2.11 highlights the the three core interfaces for PET's provider infrastructure.

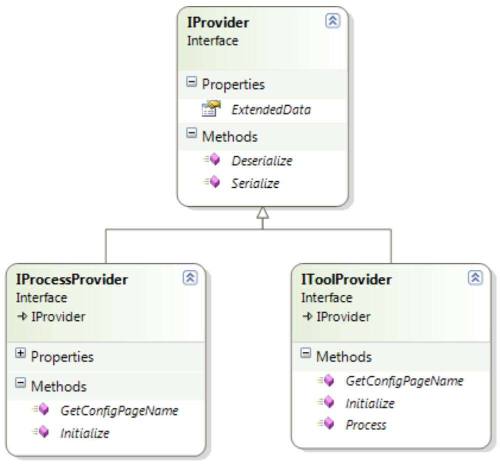


Figure 2.11.: Providers in PET

Besides the links to the data structures defined by the core model, process providers define the operations described in the following:

- `Initialize`
- `GetConfigPageName`
- `Serialize`
- `Deserialize`

`Initialize` provides the concrete process model mapping by storing the process data into the PET data structures[3]. The method `Initialize` is executed immediately after configuring the process provider, even before a tool provider is selected. After its execution the core model must be completely initialized. Refer to Sect. 2.2.3 for a detailed description of the data flow in PET.

The method `GetConfigPageName` delivers the source URI of the first configuration page of the process provider. Using configuration pages, providers support ”internal” micro workflows, which control the work of a particular provider. How configuration pages are created and integrated into the transformation workflow is explained in App. A.

The methods `Serialize` and `Deserialize` are used by the application frame to allow saving and restoring of the provider’s configuration data. As described before the data is stored in the `settings` node of the PET project file. Note that all data entered by the user should be persistent, so that there is no need for the user to provide the same data multiple times within one transformation project..

2.5. Tool Provider

Tool providers are the counterpart to process providers. They are responsible for the final transformation of the core model’s data structures into a format, the target tool can process. Just like a process provider for a certain process, tool providers are specific to a particular tool or a tool family[4].

A tool provider implements at least the methods: `GetConfigPageName`, `Serialize`, `Deserialize`, `Initialize`, and `Process`. The first three methods are similar to the methods of the process provider. The behavior of the method `Initialize` differs from its pendant as it is called just before a particular process provider is selected (the framework always initializes all available tool providers).

The method `Process` implements the concrete transformation. Here the data from the PET core model is transformed according to the needs of the target tool of the selected provider. App. B gives an example.

3 As parameter, this method takes an instance of the class *Log* that provides status information. It is recommended to use this class to get information of the transformation process, especially if errors occur.

4 A particular tool is addressed if the conversion creates an export that can only be processed by a specific tool. A tool family is supported, if the export target format is used to exchange data, e.g. Microsoft Word.

3. Reference Implementation

The described system was realized in the project *Process Enactment Tool Framework* as a reference implementation. The reference implementation consists of the PET core, the application framework and a set of process and tool providers. In detail PET's current reference implementation contains the providers for:

- V-Modell XT (process)
- Microsoft Team Foundation Server (tool)
- Microsoft Sharepoint (tool)
- Microsoft Office Word (tool)

This chapter describes all components of the reference implementation, their usage and capabilities. It does not cover work in progress. For further information, the project's web page on Codeplex[1] should be consulted.

Overview

1 http://pet.codeplex.com

3.1. Prerequisites

The Process Enactment Tool Frameworkis realized using the Microsoft .NET Framework version 3.5. The language used for the implementation is (mainly) C#. To execute the PET wizard, Windows XP, Vista or 7 must be installed. PET supports 32 Bit as well as 64 Bit operating systems, running Microsoft .NET.

Except the application frame's GUI components, all (core) components of PET (including the implemented process and tool providers) are implemented with respect to compatibility with .NET version 2.0. If required, PET can easily be ported to other operating systems such as Linux or Mac OS X using *Mono*.

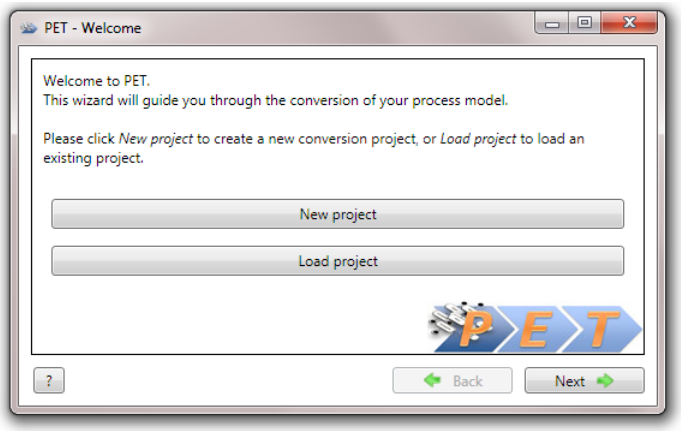
3.2. The Process Enactment Tool Framework Wizard

The main component the user interacts with is the wizard (realized in *Tum.CollabXT.Wizard*). It has been mentioned in Sect. 2.2 that this component is responsible for:

- management and rendering of the GUI
- management of the plugins
- presentation of- and navigation through the plugins' configuration pages
- controlling of the transformation workflow
- loading and storing of project data

As GUI framework we use the Windows Presentation Foundation (WPF). We chose WPF because it eases the integration of external user interface components as used by the PET plugins' configuration pages.

A screenshot of the wizard is shown in Fig. 3.1. On the first page, the user is welcomed to the wizard; two large buttons allow him/her to create a new process transformation project or to load an existing one. If an existing project is loaded, the contained settings are restored. In the next step, a concrete process provider must be selected. This is followed by the selected provider's configuration and the selection of tool providers appropriate for the intended transformation product. After their configuration the transformation is executed. On the last page of the PET wizard, the user is able to save the transformation project and to close the program.


Welcome to PET.
This wizard will guide you through the conversion of your process model.

Please click *New project* to create a new conversion project, or *Load project* to load an existing project.

Figure 3.1.: Welcome page of the PET Wizard

Each of the described steps may contain belonging wizard pages. The number of concrete wizard pages can differ according to the specific transformation workflow and the implemented fine grained provider workflow steps, which the particular providers implement (Sect. 2.2.3). Continuing the workflow is only possible after having provided all necessary inputs on a particular wizard page.

3.3. Reference Providers

Based on the PET framework several providers for processes and tools were implemented. This chapter introduces the *reference implementations*[2] and describes use cases for their application. In detail we cover the process provider for the V-Modell XT (Sect. 3.3.1) and the tool providers for Microsoft Sharepoint (Sect. 3.3.2), Microsoft Team Foundation Server (Sect. 3.3.3), and Microsoft Office Word (Sect. 3.3.4).

3.3.1. Process Provider: V-Modell XT

This process provider supports V-Modell XT users by mapping specific contents of a project-specific (tailored) V-Modell XT instance into the PET intermediate model. Currently the process provider requires a configuration file generated by the V-Modell XT Project Assistant (**.vmp*) and a complete V-Modell XT XML file, including all process structures[3].

Mapping the V-Modell XT to PET

To initialize the PET core model, the V-Modell XT elements listed in Tab. 3.1 are necessary. We see that not all (possible) elements of the V-Modell XT are necessary with regards to enactment. Consequently, the process provider omits the elements not needed, but has them available in the `ExtendedData` field of the respective process elements. To get a detailed overview over the V-Modell XT elements, refer [TK09].

Element	Description	Mapped by
Process Modules	Process modules contain all structural necessary elements. Usually not all process modules defined in the process are required. The set is defined by the tailoring profile stored in the **.vmp* file.	–
Work Products	Work products are used to get the results (artifacts) of the process, based on the process modules given by the tailoring profile.	`Artifact`
Topics	Topics, if defined, structure and refine work products.	`Topic`
Decision Gates	Decision gates model milestones. Which are necessary results from the tailoring profile.	`Milestone`
Activities	*See work products.*	`Activity`
Tasks	*See topics*	`Task`
Disciplines	Disciplines are used to structure work products. Logical grouping is used for further work in the tools, e.g. grouping work products on the user interface to easy selection.	`Discipline`
Roles		`Role`
Associations	Associations build the structure of the process by coupling the elementary process elements. Possible types of associations are defined in the PET core model (see Sect. 2.3.9).	–

Table 3.1.: V-Modell XT model elements required for the mapping

Inside the Process Provider. To get a valid mapping it is important to check which process modules are relevant for a project (see Tab. 3.1). The method `LoadVBs` (class `VModellXTProcessProvider`) performs this check. `LoadProducts` then loads the relevant modules' work products and maps them to the class `Artifact`. Afterwards, the method `LoadDecisionGates` loads the relevant decision gates from the pre-planned configuration in the **.vmp*-file. They are the basis for instantiating the class `Milestone` of the core model.

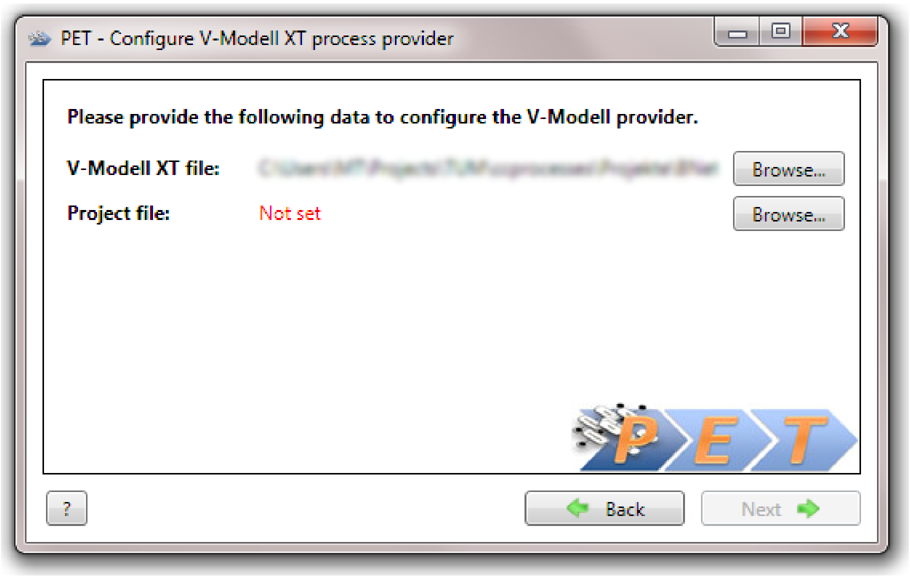
2 The reference implementations are bundled with the framework and packaged for download at Codeplex.
3 For future work an integrated tailoring mechanism is planned to make the PP independent from the Project Assistant outputs.

Hint: The information of the pre-planned decision gates combined with the process information given by the *project execution strategy* that is selected by the tailoring, is the basis for e.g. deriving *iteration paths* as necessary for the TFS provider (Sect. 3.3.3).

After reading the decision gates, the process provider loads the roles involved with the process. Together with the already loaded decision gates, first structure information is loaded by the method `LoadDependencies`. Collecting all association goes beyond the simple relation between work products and decision gates. The V-Modell XT is rich of association, which are completely covered by the PET core model. Thus, work products and activities are also related by associations, project plan including concrete milestones and assigned activities, and also corresponding roles, can be derived.

Configuration. PET provides a framework allowing plugins to provide their own user interface parts. The V-Modell XT process provider provides an XAML user interface. It allows the user to configure[4] the used V-Modell XT. Furthermore the project configuration file (**.vmp*) must be provided. Fig. 3.2 shows the provider's configuration page.

Note, that as far as the process XML-file is conform to the meta-model 1.3 any V-Modellderivate can be used with PET.


PET - Configure V-Modell XT process provider

Please provide the following data to configure the V-Modell provider.

V-Modell XT file:	[redacted]	Browse...
Project file:	Not set	Browse...

Figure 3.2.: V-Modell XT process provider configuration page

Special Hints. To avoid far-reaching changes in the code of the plugin in case of a V-Modell XT structure change, all relevant XPath queries that gather the data are located in an extra resource file (see Fig. 3.3). If e.g. only the meta model changes in a way that elements are re-located in the model, the XPath queries can easily be updated.

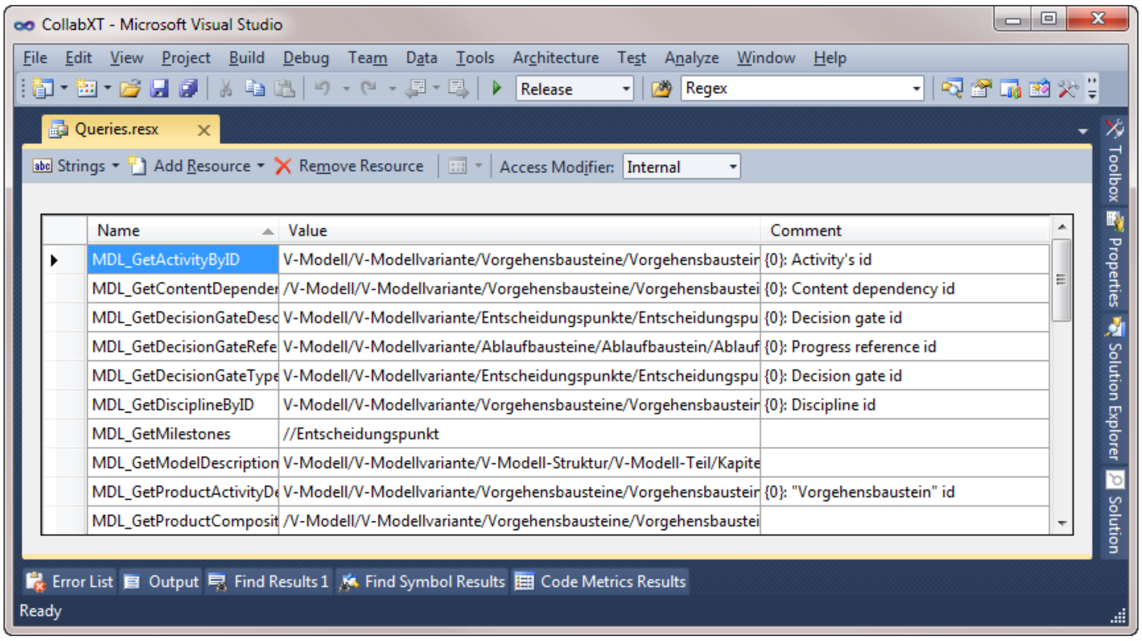
3.3.2. Tool Provider: SharePoint

The tool provider for Microsoft SharePoint generates a team website based on the input process model. The team website is meant to be a workspace for projects that have to follow the process given by the process provider. The target audience of a PET-generated SharePoint team website are the less development-focused roles in a project, especially the management.

The SharePoint tool provider generates content of the process model into the website and links up the content items according to the structure of the input process.

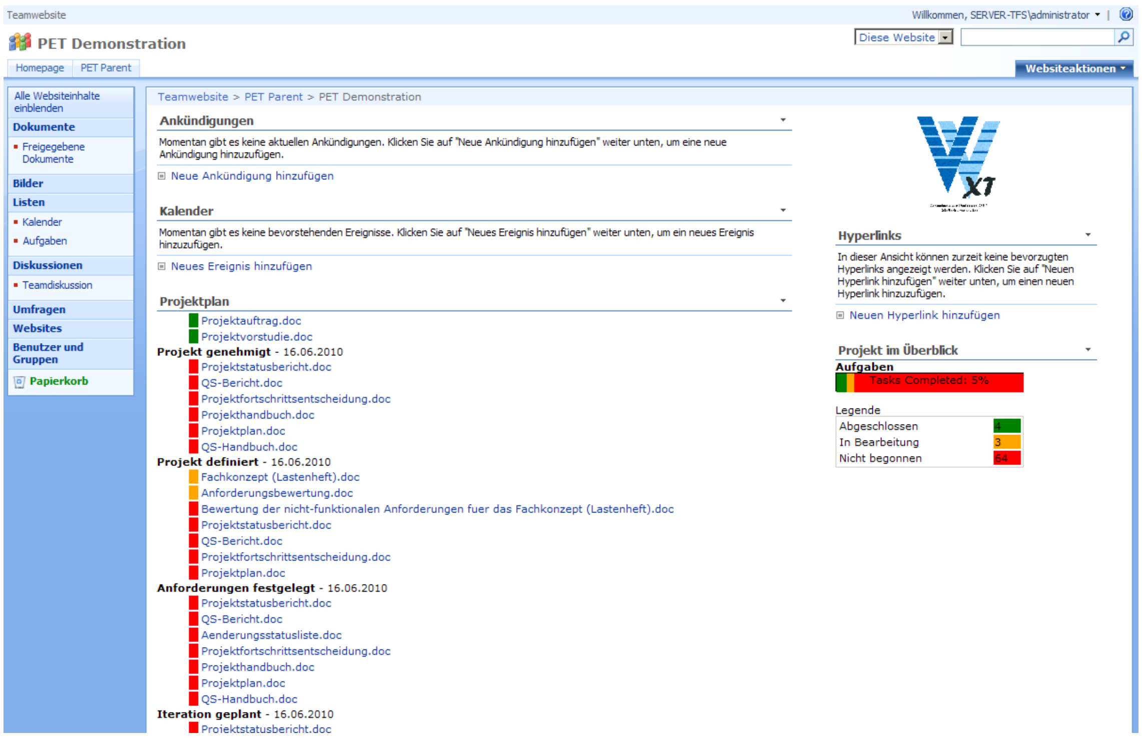
Unlike the tool provider for Microsoft Team Foundation Server (see Sect. 3.3.3), the SharePoint tool provider directly uses the API of the target tool. This especially means that the tool provider has to run on the target server where the SharePoint portal is going to be created.

4 To avoid multiple user inputs, the plugin stores the recent path for the V-Modell XT XML-file.



Name	Value	Comment
MDL_GetActivityByID	V-Modell/V-Modellvariante/Vorgehensbausteine/Vorgehensbaustein	{0}: Activity's id
MDL_GetContentDepender	/V-Modell/V-Modellvariante/Vorgehensbausteine/Vorgehensbaustei	{0}: Content dependency id
MDL_GetDecisionGateDesc	V-Modell/V-Modellvariante/Entscheidungspunkte/Entscheidungspu	{0}: Decision gate id
MDL_GetDecisionGateRefe	V-Modell/V-Modellvariante/Ablaufbausteine/Ablaufbaustein/Ablauf	{0}: Progress reference id
MDL_GetDecisionGateType	V-Modell/V-Modellvariante/Entscheidungspunkte/Entscheidungspu	{0}: Decision gate id
MDL_GetDisciplineByID	V-Modell/V-Modellvariante/Vorgehensbausteine/Vorgehensbaustein	{0}: Discipline id
MDL_GetMilestones	//Entscheidungspunkt	
MDL_GetModelDescription	V-Modell/V-Modellvariante/V-Modell-Struktur/V-Modell-Teil/Kapite	
MDL_GetProductActivityDe	V-Modell/V-Modellvariante/Vorgehensbausteine/Vorgehensbaustein	{0}: "Vorgehensbaustein" id
MDL_GetProductComposit	/V-Modell/V-Modellvariante/Vorgehensbausteine/Vorgehensbaustei	

Figure 3.3.: V-Modell XT queries resource file in Visual Studio 2010



Ankündigungen

Momentan gibt es keine aktuellen Ankündigungen. Klicken Sie auf "Neue Ankündigung hinzufügen" weiter unten, um eine neue Ankündigung hinzuzufügen.

Neue Ankündigung hinzufügen

Kalender

Momentan gibt es keine bevorstehenden Ereignisse. Klicken Sie auf "Neues Ereignis hinzufügen" weiter unten, um ein neues Ereignis hinzuzufügen.

Neues Ereignis hinzufügen

Projektplan

- 🟩 Projektauftrag.doc
- 🟩 Projektvorstudie.doc

Projekt genehmigt - 16.06.2010
- 🟥 Projektstatusbericht.doc
- 🟥 QS-Bericht.doc
- 🟥 Projektfortschrittsentscheidung.doc
- 🟥 Projekthandbuch.doc
- 🟥 Projektplan.doc
- 🟥 QS-Handbuch.doc

Projekt definiert - 16.06.2010
- 🟧 Fachkonzept (Lastenheft).doc
- 🟧 Anforderungsbewertung.doc
- 🟧 Bewertung der nicht-funktionalen Anforderungen fuer das Fachkonzept (Lastenheft).doc
- 🟥 Projektstatusbericht.doc
- 🟥 QS-Bericht.doc
- 🟥 Projektfortschrittsentscheidung.doc
- 🟥 Projektplan.doc

Anforderungen festgelegt - 16.06.2010
- 🟥 Projektstatusbericht.doc
- 🟥 QS-Bericht.doc
- 🟥 Aenderungsstatusliste.doc
- 🟥 Projektfortschrittsentscheidung.doc
- 🟥 Projekthandbuch.doc
- 🟥 Projektplan.doc
- 🟥 QS-Handbuch.doc

Iteration geplant - 16.06.2010
- 🟥 Projektstatusbericht.doc

Hyperlinks

In dieser Ansicht können zurzeit keine bevorzugten Hyperlinks angezeigt werden. Klicken Sie auf "Neuen Hyperlink hinzufügen" weiter unten, um einen neuen Hyperlink hinzuzufügen.

Neuen Hyperlink hinzufügen

Projekt im Überblick

Aufgaben

Tasks Completed: 5%

Legende

Abgeschlossen	🟩 4
In Bearbeitung	🟧 3
Nicht begonnen	🟥 64

Figure 3.4.: PET SharePoint portal home

The SharePoint site template.

The SharePoint tool provider expects certain WebParts and SharePoint site templates to be installed on the server. Before running the tool provider for the first time, these have to be installed on the server [5]. The prerequisites are packaged in a SharePoint solution. It has the following contents:

- A localization assembly. This assembly contains language resources so that PET can create a portal on both a German and an English SharePoint server.

5 Detailed instructions about the installation of the prerequisites can be found here: `http://pet.codeplex.com/wikipage?title=Getting%20Started%20with%20SharePoint`

- The *Project Plan* web part: This web part shows the milestone plan that has been planned with the V-Modell XT project assistant. Fig. 3.4 depicts the web part in the middle of the website. The milestones themselves are displayed in black. Above each milestone is a list of the work products that have to be completed for the given milestone (according to the input process model).
- The *Project Overview* web part: This web part displays a colored progress bar showing the number of tasks in the task list that are completed, in progress and not started. A screenshot of this web part can be seen in Fig. 3.4.
- The V-Modell XT site template. It contains for example the V-Modell XT logo that is featured on the portal home page.

Structure of the generated site.

The tool provider for SharePoint was originally developed only for the V-Modell XT. Thus the structure of a generated portal is in large parts inspired by the structure of the contents of the V-Modell XT. The structure of a PET-generated portal is depicted in Fig. 3.5.

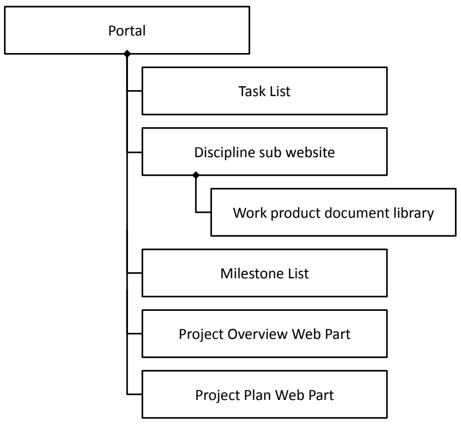


Figure 3.5.: Structure of a PET-generated portal

The mapping of process model contents onto the SharePoint portal works as follows:

Disciplines: A discipline is an element of the intermediate model used to group process model contents. The current implementation of the SharePoint tool provider creates a sub website below the main portal for each discipline. The discipline's description is generated into the title area of the sub website.

Work Products: The site template used for the discipline sub website contains a predefined SharePoint list of type *Document Library*. This library is filled with the work product templates of the work products belonging to the given discipline. The generated portal will contain a sub website for each discipline, each containing a document library with the work product templates for the work products of that discipline. The default SharePoint document library is extended by two additional fields: The first one lets the user specify the work product state from a set of predefined values. These are *In Processing*, *Submitted*, and *Finished* (inspired by the default product states of the V-Modell XT). The product state field may eventually be used to attach a workflow to the document library. The second field is of the SharePoint type `MultiChoice` and indicates to which milestones the work product has to be submitted to.

Activities: In the current implementation, the tasks are managed in one list. Theoretically, the tasks could be managed at the discipline-level – like work products. However, the fragmentation of the task list would make reporting over all tasks unnecessarily complex. Compared to the SharePoint default task list, the task list in a PET-generated portal has one additional field to reference the work product associated with the activity (or task).

Roles: Roles are not mapped in the current implementation. A possible mapping candidate are SharePoint *roles* or user groups.

Milestones: The tool provider creates a custom list in the portal to manage the milestones and planning information. Table 3.2 shows the column definition of that list.

Column Name	Data type	Remark
Title	`Text`	The name of the milestone.
Date	`DateTime`	The due-date of the milestone
Number	`Integer`	The number of the milestone. The number is used to order the milestones.
Description	`Text`	The description of the milestone

Table 3.2.: Milestones list definition

3.3.3. Tool Provider: TFS

The tool provider for Microsoft Team Foundation Server (TFS) creates a complete and valid process template for TFS 2005/2008[6]. The provider's output can be imported into a running TFS instance and serve as template for new TFS projects.

Generating the TFS template.

The TFS tool provider requires a template – a so called *meta template* – for the output generation. The meta template contains elementary types and structures; during the process transformation it is filled with contents from the PET core model. So the meta template already defines the basic structures regarding folders, (special) files (Fig. 3.6) and so on. The meta template is located in the subfolder `tpl` of the plugin directory.

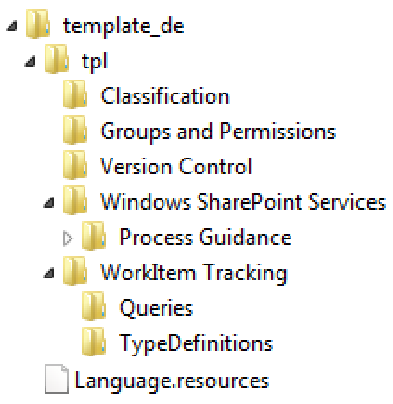


Figure 3.6.: Basic folder structure of the meta template (no files shown).

Tab. 3.3 summarizes the contents of the meta templates. Readers who are familiar with TFS see that the plugin does not pre-define reports. The tool provider intentionally leaves this topic open for the organization-specific tailoring of the meta template (which includes relevant reports).

Element	Description
Classification	Structure of the project
Groups and Permissions	Definition of roles and information for the security management
Version Control	Setup information for TFS version control
Windows Sharepoint Services	Contains all files belonging to the process documentation (stored on the TFS-integrated Sharepoint server)
WorkItem Tracking	Contains all type definitions for the work items relevant in the project; basic queries over the types are defined here

Table 3.3.: Contents of the tpl subfolder

6 Due to process template incompatibilities and new features, such as hierarchical work items, a new tool provider will be created explicitly for TFS 2010.

A modern Babel. . .

At this point there is something important to know: The tool provider for TFS actually contains two meta templates: one for a German and one for an English TFS. The meta template used for the export must match the server's *language* version. TFS has hard requirements regarding the language of a template. A German template will not install on a running English TFS and vice versa. To define a suitable mapping, the file *Language.resources* defines the necessary language mappings.

Inside the Provider. Simply put a TFS process template is a collection of files – usually XML files. So creating a valid process template is principally done by modifying XML structures. In the following, this task is described using the file *WorkItems.xml* as an example. This file contains initialized work items based on the process.

The first step is to copy the whole meta template to the output directory. The tool provider in the following fills the XML section *tasks/task[@id='WIs']*, which contains the initially instantiated work items. Amongst others the set of relevant work items contains milestones, tasks, and artifacts that are available at the project's start-up. Each instantiation is basically an insertion of a *WI*-node into the XML structure. Depending on the concrete work item type (WIT), several modifications are required. Listing 3.1 shows the data structure for a milestone[7].

```
<WI type="Entscheidungspunkt">
  <FIELD refname="System.Title" value="Projekt␣genehmigt" />
  <FIELD refname="System.Description"
        value="In␣dem␣Entscheidungspunkt␣Projekt␣genehmigt␣..." />
  <FIELD refname="System.State" value="Geplant" />
  <FIELD refname="System.IterationPath"
        value="$$PROJECTNAME$$\00␣Projektinitialisierung" />
  <FIELD refname="VMXT.id" value="de21fb30c4aec0" />
  <FIELD refname="VMXT.Entscheidungspunkttyp"
        value="Projekt␣genehmigt" />
  <FIELD refname="Microsoft.VSTS.Scheduling.FinishDate"
        value="07/11/08" />
</WI>
```

Listing 3.1: A concrete milestone work item data structure

The names of the fields are defined in the WI description and referred by *refname*. When providing a concrete *value* it is necessary to exactly follow the respective schema rules, as TFS is very strict in parsing the template's content. If the data is not matching its requirements, the template is be rejected.

Example: For example it is necessary to take care of limitations regarding the length of strings.

Work Item Types

The tool provider for TFS contains standard definitions for selected work item types (as already discussed in [KK08b]). However, due to technical improvements, new and extended work item types were developed. In this section we give a brief overview over the current work item types, their data structure and the mapping used by the provider.

WIT: Product. Tab. 3.4 shows the fields of the *Product* work item type. Note, that fields that are marked with an asterisk (*) are not modified by the provider. Products in TFS are representatives for work products in a project. This work item type is designed for controlling aspects.

Besides creating the work item nodes for all initial products, the TFS tool provider also modifies the belonging work item type description itself. *Product* work items may represent products of different types (see Tab. 3.4). To represent this product type as a dropdown list (instead of a text field) in the WI user interface it is necessary to register all possible types in advance. The tool provider does this by correctly filling in the allowed values for the field *VMXT.ProduktTyp*.

7 Basis for the example is a German V-Modell XT instance.

WI field	Content
System.Title	*Name* of the product
System.Description	Product's *Description* property
System.History	History information
System.AreaPath (*)	Logical area of the product
System.IterationPath (*)	Iteration in which the product is created
System.State (*)	Workflow state of the product
System.Reason (*)	Reason the product is in its current state
System.AssignedTo (*)	User responsible for the product
Microsoft.VSTS.Scheduling.FinishDate (*)	Date the product is scheduled to be finished
Microsoft.VSTS.Common.ClosedDate (*)	Date the product was closed
Microsoft.VSTS.Common.ClosedBy (*)	User that closed the product
VMXT.id	Identifier of the product
VMXT.ProduktTyp	Content of the *TypeName* field
VMXT.Erzeugung	Fixed value: "Initial" (as only initial products are represented in the process template)
VMXT.Assessment (*)	Documentation of the test results

Table 3.4.: Content of the fields of Product work items.

WIT: Decision Gate. In Tab. 3.5 the fields of *Decision Gate* work items are listed and explained. For reasons of readability, the *System* and *Microsoft.VSTS* fields that have already been mentioned in Tab. 3.4 are omitted; however, if a field's meaning or content differs from its equivalent in the Product work item it is listed again. This schema is used for all further work item types.

WI field	Content
System.State	Fixed value: "Planned"
System.IterationPath	TFS iteration path – processed based on the *ExtendedData* field *DevPhase*
Microsoft.VSTS.Scheduling.FinishDate	Value of the *ScheduledDate* field, planned date this decision gate is finished
VMXT.id	Identifier of the decision gate
VMXT.Entscheidungspunkttyp	Decision gate type – based on the *ExtendedData* field *TypeName* if it exists
VMXT.Assessment (*)	Results of the decision gate

Table 3.5.: Content of the fields of Decision Gate work items.

Just like the Product type definition, the Decision Gate type definition is modified in the template generation process. The TFS tool provider sets the allowed values for the Decision Gate field *VMXT.Entscheidungspunkttyp* to resemble the decision gate types of the input process.

WIT: Activity. Tab. 3.6 describes the fields of the *Activity* work item type. Again, fields already explained before are not contained.

WI field	Content
System.State	Fixed value: "Planned"
System.IterationPath	TFS iteration path – based on the belonging milestone's (decision gate's) iteration path
Microsoft.VSTS.Scheduling.FinishDate	Planned date this activity is finished – based on the belonging milestone's (decision gate's) finish date
VMXT.id	Identifier of the activity
VMXT.EPRef	Identifier of the milestone (decision gate) the activity belongs to – determined by the relation
VMXT.PRef	Artifact (product) the activity belongs to (e.g. creating it) – determined by the relation

Table 3.6.: Content of the fields of Activity work items.

WIT: Task. In Tab. 3.7 the fields of the work item type *Task* are given. Note again, that fields listed in Tab. 3.4 are omitted.

WI field	Content
System.IterationPath	TFS iteration path – based on the belonging activity's iteration path
Microsoft.VSTS.Common.Priority (*)	The task's priority
Microsoft.VSTS.Common.Severity (*)	Influence of the task on the project
Microsoft.VSTS.Scheduling.RemainingWork (*)	Remaining hours needed to complete the task
Microsoft.VSTS.Scheduling.CompletedWork (*)	Hours already worked on the task
Microsoft.VSTS.Scheduling.StartDate (*)	Planned date this task is started
Microsoft.VSTS.Scheduling.FinishDate	Planned date this task is finished – extracted from the belonging activity's finish date
Microsoft.VSTS.Scheduling.TaskHierarchy (*)	Task context
VMXT.ARef	Identifier of the activity the task belongs to – determined by the relation
VMXT.Discipline	Discipline the task belongs to – determined by the relation
VMXT.Estimate (*)	Hours needed to complete the task
VMXT.Blocked (*)	Set if the work on the task is blocked

Table 3.7.: Content of the fields of Task work items.

The value of the field *VMXT.Discipline* is selected from a fixed set of disciplines. PET's TFS tool provider modifies the Task work item type definition file to enforce this behavior.

Provider Configuration

It has been mentioned before that the tool provider for TFS needs a meta template to generate the final process template. Consequently, the first step of the provider configuration (Fig. 3.7) is to select a valid meta template. Additionally the provider allows the user to enter a path, where the process documentation (preferably in HTML) can be found. This documentation is merged with the one contained in the meta template. It is registered on the TFS upon instantiation of the process template.

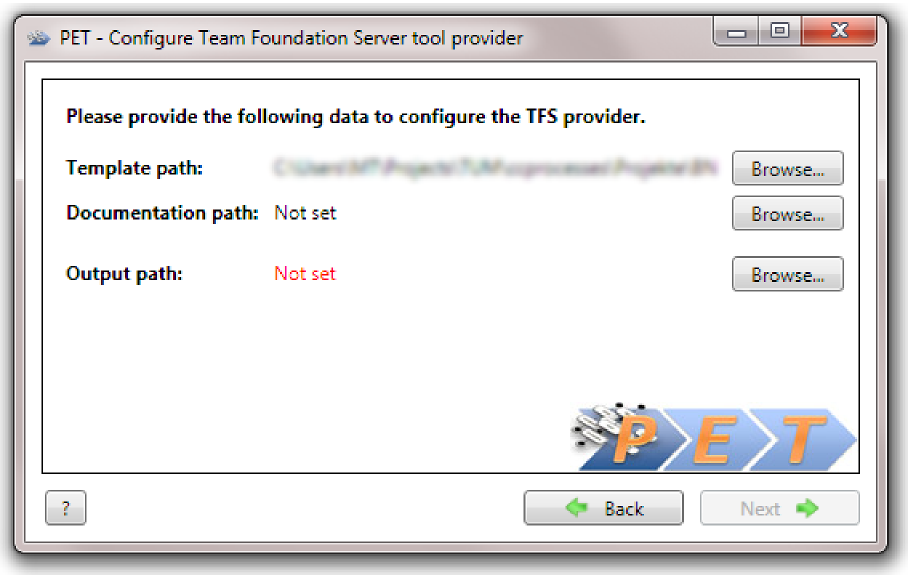
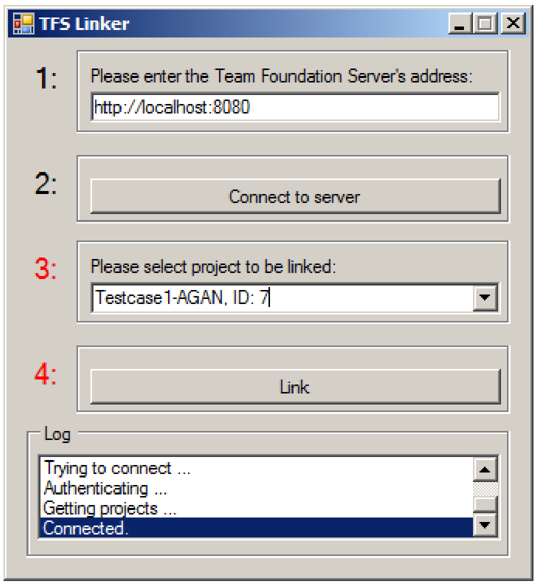

PET - Configure Team Foundation Server tool provider
Please provide the following data to configure the TFS provider.
Template path:
Browse...
Documentation path: Not set
Browse...
Output path: Not set
Browse...
PET
?
Back
Next

Figure 3.7.: Configuration page of the Team Foundation Server tool provider.

Work Item Linking

The tool provider for TFS suffers a great problem with process templates. It is not possible to declaratively define complex structures of a project within a process template. This means, concrete associations (in terms of TFS: *links*) cannot be created through the template, as the required object identification is only available *after* a project's instantiation. Hence, the processes and the PET core model contain rich information about process structures, and also TFS can handle complex structures, there is no possibility to declare relations in the process template.

To improve the process template, all PET TFS work item types contain meta data (fitted for using them with V-Modell XT). The meta data contains the necessary information to provide a simple, initial ”runtime-support” by being able to connect the loosely coupled work item (instances) on the server. A separate tool (*TfsLinker.exe*) realizes the runtime linking based on the modeled associations. The linker's user interface is shown in Fig. 3.8.


TFS Linker
1:
Please enter the Team Foundation Server's address:
http://localhost:8080
2:
Connect to server
3:
Please select project to be linked:
Testcase1-AGAN, ID: 7
4:
Link
Log

```
Trying to connect ...
Authenticating ...
Getting projects ...
Connected.
```

Figure 3.8.: Screenshot of the PET TFS Linker.

Firstly, the TFS's address must be entered (including *http://*); the *Connect to server* button then establishes the connection and authenticates the user. Once the linker is connected, the user can select the project for which the links are to be created. The *Link* button finally starts the linking process. Note, that the linker is no part of the PET; it is only provided with the TFS provider for user convenience. Consequently, the technical details of the program are not discussed here.

3.3.4. Tool Provider: Microsoft Office Word

The last tool provider, contained in the reference implementation of PET is the *Microsoft Office Word* provider. It allows the generation of document templates in the *Office Open XML* (OOXML) format. The generated documents contain the respective artifact's section structure (headings) as well as the sections' predefined texts and formating elements as given by the process model.

As basis for the document template generation the tool provider uses the artifact-, topic- and role information given by the PET intermediate model as well as the associations between these elements. Additionally, the user may select a template text definition file, which must be in the *V-Modell XT template text XML* format.

The Master Document

The created documents' layout is given by *generation templates* in the OOXML format that should contain placeholders specific to the tool provider. Using OOXML files as both document generation templates and the generation result has the advantages that

- arbitrary document and content formating is possible, and
- Microsoft Word can be used to edit the document templates.

Top-level Placeholders. The tables 3.8 and 3.9 list and describe the possible placeholders. Note, that all placeholder names have the prefix *"PRODUKT."*; for layout reasons these prefixes are omitted in the tables.

Placeholder	Description
GRUPPE	Discipline of the artifact
NAME	The artifact's name
PROJEKTBEZEICHNUNG	Name of the project
VERANTWORTLICHER	Role responsible for the artifact
MITWIRKENDE (*)	Region for roles involved in the creation of the artifact
ATTRIBUTE (*)	Region for attributes of the artifact
ERZEUGUNGEN (*)	Region for properties related to the creation of the artifact
SINNUNDZWECK	The artifact's purpose
THEMEN (*)	Region for the topics of the artifact; in most cases the main part of the document
ABHAENGIGKEITEN (*)	Region for structural and content-related dependencies of the artifact

Table 3.8.: Top-level document generation template placeholders.

Region Placeholders. Placeholders that are marked with an asterisk (*) are so-called *region placeholders*. Region placeholders must be used in pairs to mark the beginning and the end of a region in the template. To do so they must be appended the respective suffixes *".BEGINN"* (beginning) and *".ENDE"* (end). Within a region special placeholders, listed in Table 3.9, are valid.

Subtopic placeholders (prefix *"PRODUKT.THEMA.SUB"*) may be nested. The level marker *"[N]"* used in Table 3.9 must be replaced by the level of the respective subtopic placeholder. E.g. the first subtopic region is surrounded by the placeholders *"PRODUKT.THEMA.SUB1.BEGINN"* and *"PRODUKT.THEMA.SUB1.ENDE"*. Within the subtopic nesting hierarchy it is *not* permitted to skip levels.

Region placeholder	Placeholder	Description
MITWIRKENDE	MITWIRKEND.NAME	Name of the involved role
ATTRIBUTE	ATTRIBUT.NAME	Name of the artifact property
ERZEUGUNGEN	ERZEUGUNG.NAME	Name of the creation-related property
ERZEUGUNGEN	ERZEUGUNG .QUELLPRODUKTE (*)	Region for associated, creation-related artifacts
ERZEUGUNGEN .QUELLPRODUKTE	ERZEUGUNG.ANDERESPROD .NAME	Name of the creation-related artifact
THEMEN	THEMA.NAME	Name of the topic
THEMEN	THEMA.BESCHREIBUNG	Topic description
THEMEN	THEMA.MUSTERTEXTINHALT (*)	Region for topic template text
THEMA .MUSTERTEXTINHALT	THEMA.MUSTERTEXTINHALT	Topic template text
THEMEN	THEMA.SUB[n] (*)	Region for sub-topic
THEMA.SUB[n]	THEMA.SUB[n].NAME	Sub-topic name
THEMA.SUB[n]	THEMA.SUB[n].BESCHREIBUNG	Sub-topic description
THEMA.SUB[n]	THEMA.SUB[n] .MUSTERTEXTINHALT (*)	Region for sub-topic template text
THEMA.SUB[n] .MUSTERTEXTINHALT	THEMA.SUB[n] .MUSTERTEXTINHALT	Sub-topic template text
THEMA.SUB[n]	THEMA.SUB[n+1] (*)	Region for sub-sub-topic; see *subtopic placeholders*
ABHAENGIGKEITEN	ABHAENGIGKEIT.NAME	Name of the dependency
ABHAENGIGKEITEN	ABHAENGIGKEIT .BESCHREIBUNG	Description of the dependency
ABHAENGIGKEITEN	ABHAENGIGKEIT .ANDEREPRODUKTE (*)	Region for dependent artifacts
ABHAENGIGKEIT .ANDEREPRODUKTE	ABHAENGIGKEIT. ANDEREPRODUKT.PRODUKT	Name of the dependent artifact

Table 3.9.: Conditional document generation template placeholders.

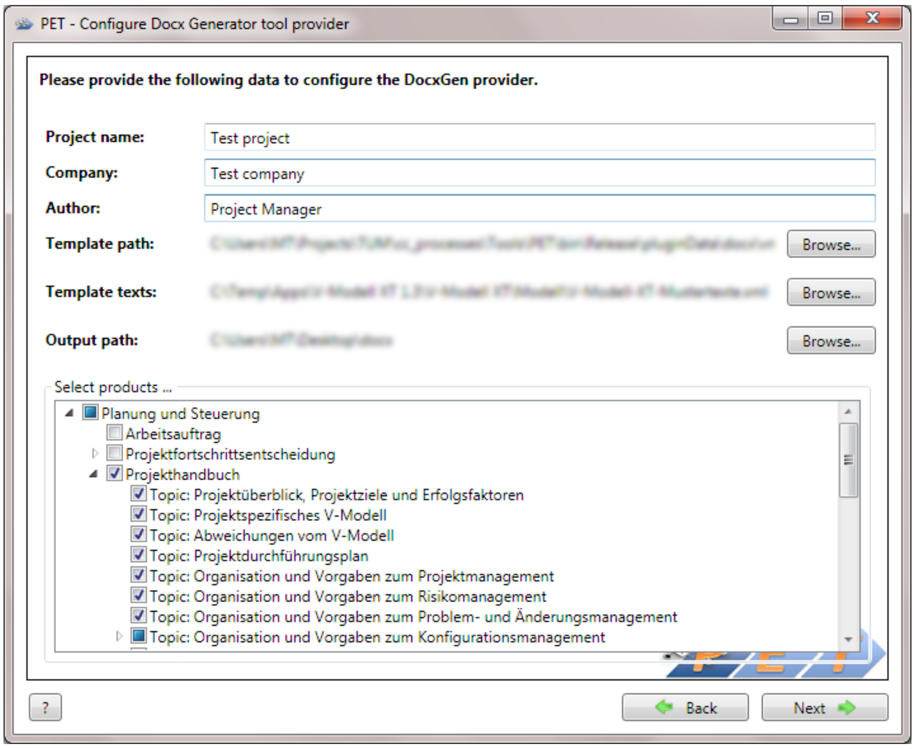
Provider Configuration

Fig. 3.9 shows the configuration page of the tool provider. To allow the generation process, the template and output paths must be entered. Additionally the artifacts to be created must be selected.

The user may specify the template texts' path, so that they can be inserted into the created artifacts. If template texts are available, they also appear in the artifact selection tree. Template texts marked as "default" by the model are initially selected.

Furthermore, the project name, the company and the artifact author can be entered. If this is done, these information are put into the generated artifacts at the respective placeholders as well as in the documents' meta data.

Nice to know for customization and application. The tool provider for Microsoft Word is very flexible. It supports the project-specific selection of a master template (the comparable function of the V-Modell XT reference tool only supports process-specific templates, one per process variant). So, templates can be provided in different languages, styles etc. Another aspect, important for the user, is the fact that the PET provider supports a "single document export". Single document export means the possibility of exporting just the one template that is currently needed. The comparable tools export templates also respecting their attributes, which in case of the V-Modell XT means that the *initial* work product templates are exported each time by overwriting eventually exiting documents. Also if interest is the fact that the PET provider analyses the attributes, too.


PET - Configure Docx Generator tool provider

Please provide the following data to configure the DocxGen provider.

Project name:	Test project	
Company:	Test company	
Author:	Project Manager	
Template path:	[redacted]	Browse...
Template texts:	[redacted]	Browse...
Output path:	[redacted]	Browse...

Select products ...

- Planung und Steuerung
  - ☐ Arbeitsauftrag
  - ☐ Projektfortschrittsentscheidung
  - ☑ Projekthandbuch
    - ☑ Topic: Projektüberblick, Projektziele und Erfolgsfaktoren
    - ☑ Topic: Projektspezifisches V-Modell
    - ☑ Topic: Abweichungen vom V-Modell
    - ☑ Topic: Projektdurchführungsplan
    - ☑ Topic: Organisation und Vorgaben zum Projektmanagement
    - ☑ Topic: Organisation und Vorgaben zum Risikomanagement
    - ☑ Topic: Organisation und Vorgaben zum Problem- und Änderungsmanagement
    - Topic: Organisation und Vorgaben zum Konfigurationsmanagement

Figure 3.9.: Microsoft Office Word tool provider configuration page.

In the context of the V-Modell XT initial work products are generated as "real", ready-to-work documents (*.docx), while non-initial ones are provides as document templates (*.dotx).

Hint: In 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.

A. How To: Implement a Process Provider

The following tutorial outlines the basic steps to implement a simple process provider. The tutorial will not show how to map a complete process model. The focus is the plugin development for the Process Enactment Tool Framework.

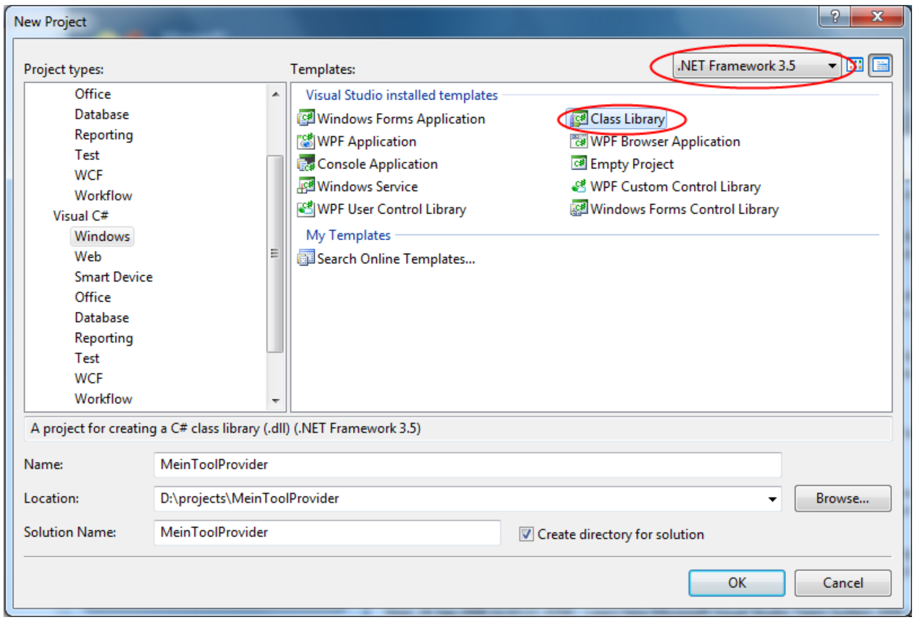
Prerequisites. Before starting with the development of the process provider itself, PET that contains the necessary libraries has to be downloaded. PET can be downloaded from Codeplex[1]. After extracting the downloaded ZIP-file, the wizard (`Tum.CollabXT.Wizard.exe`) can be started. It requires the .NET Framework 3.5, so this should be installed before attempting to start PET. If the .NET Framework 3.5 is not installed, nothing will happen when trying to start `Tum.CollabXT.Wizard.exe` (there will be no error message).

A.1. The Scenario

The process provider developed in this tutorial is a minimal example. It does not have any practical purpose and most tool providers will not produce sensible results with this process provider. It does however show the most important concepts necessary to implement a real process provider.

A.2. Preparing the process provider Project

Step 1. Both tool providers and process providers are encapsulated in .NET assemblies. The first step therefore is to create a new *Windows Class Library* project in Visual Studio. As shown in Fig. A.1, the target framework should be .NET 3.5.



.NET Framework 3.5

A project for creating a C# class library (.dll) (.NET Framework 3.5)

Name: MeinToolProvider

Location: D:\projects\MeinToolProvider

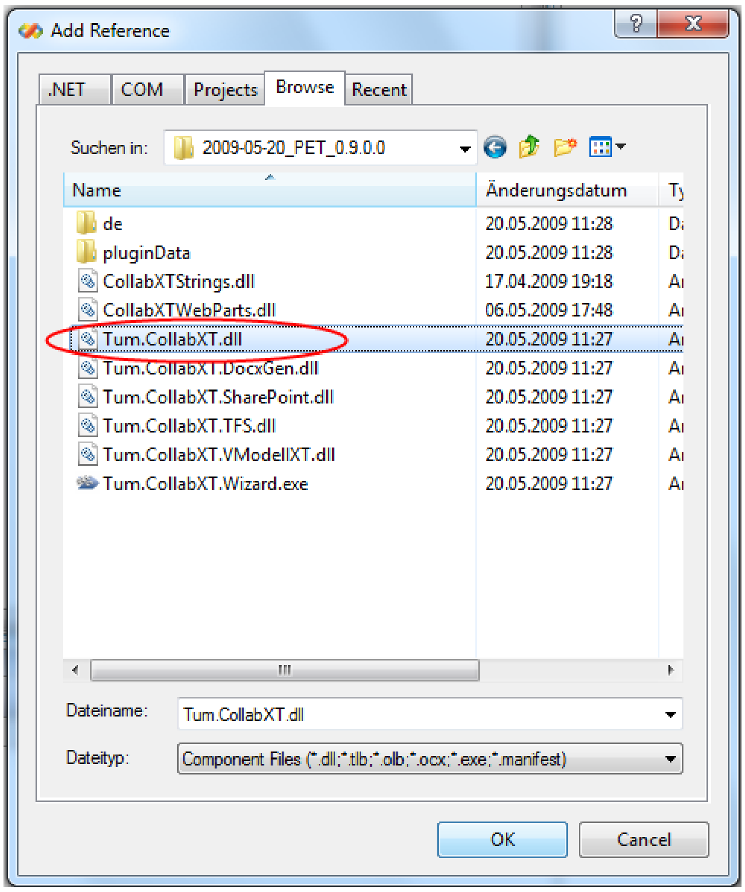
Solution Name: MeinToolProvider

[✓] Create directory for solution

Figure A.1.: Creating a new project in Visual Studio

1 `http://pet.codeplex.com`

Step 2. To make sure that the new assembly can be found by the PET framework, its output directory should be changed to the directory of PET. In Visual Studio 2008, this can be done under *Projects → [Project name] Properties ... → Build → Output path.*



Suchen in: 2009-05-20_PET_0.9.0.0

Name	Änderungsdatum	Ty
de	20.05.2009 11:28	D
pluginData	20.05.2009 11:28	D
CollabXTStrings.dll	17.04.2009 19:18	A
CollabXTWebParts.dll	06.05.2009 17:48	A
Tum.CollabXT.dll	20.05.2009 11:27	A
Tum.CollabXT.DocxGen.dll	20.05.2009 11:27	A
Tum.CollabXT.SharePoint.dll	20.05.2009 11:27	A
Tum.CollabXT.TFS.dll	20.05.2009 11:27	A
Tum.CollabXT.VModellXT.dll	20.05.2009 11:27	A
Tum.CollabXT.Wizard.exe	20.05.2009 11:27	A

Dateiname: Tum.CollabXT.dll

Dateityp: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

Figure A.2.: Reference the Process Enactment Tool Framework library

Step 3 – (initial coding). The next step is to reference the assembly `Tum.CollabXT.dll` to be able to use the plugin-interface of PET and the intermediate model. This can be done by opening the *Add Reference* dialog in Visual Studio and navigating to the directory where the PET ZIP-file has been extracted to (see Fig. A.2).

Now the class file `Class1` added by the project template should get a more meaningful name – for example `MyProcessProvider`. It should inherit from the interface `IProcessProvider`. The resulting code should look similar to listing A.1.

```
using System;
using System.Collections.Generic;
using Tum.CollabXT;

namespace HowToPP
{
    public class MyProcessProvider : IProcessProvider
    {
        public List<IActivity> Activities
        { get { throw new NotImplementedException(); } }
        // [...]
        public List<ITopic> Topics
        { get { throw new NotImplementedException(); } }

        public string Name
        { get { throw new NotImplementedException(); } }

        public string Description
        { get { throw new NotImplementedException(); } }
        public void Initialize(Log log)
        { throw new NotImplementedException(); }

        public string GetConfigPageName()
        { throw new NotImplementedException(); }

        public Dictionary<string, object> ExtendedData
        { get { throw new NotImplementedException(); } }

        public void Serialize(System.Xml.XmlNode outputParentNode)
        { throw new NotImplementedException(); }

        public void Deserialize(System.Xml.XmlNode inputParentNode)
        { throw new NotImplementedException(); }
    }
}
```

Listing A.1: Skeleton of MyProcessProvider

The interface to implement is shown in figure A.3.

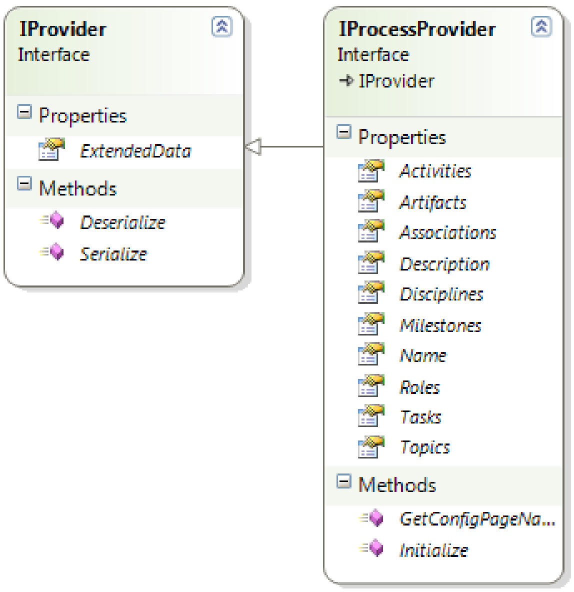


Figure A.3.: The IProvider interface

Both process providers and tool providers inherit from the general provider interface `IProvider` (see Sect. 2.3). It contains two methods that have to be implemented:

- `Serialize`: This method will be called by the framework and allows to save provider-specific configuration data to the PET project file.
- `Deserialize`: This method is the complement to `Serialize`. Provider-specific data can be loaded from the project file with this method.

For an example implementation of those two methods, see the Sect. B.6. The interface `IProcessProvider` has two more methods that will have to be implemented:

- `Initialize`: This method can be used to initialize the process provider. A reference to the intermediate model and to a logger object are passed in. The `Log` object can be used to output status information. These will not be displayed in the user interface but can be examined in the *log.txt* file which is created in the program directory.
- `GetConfigPageName`: This method is called by the framework to get the name of the configuration page of this process provider. It is used by the framework to correctly add the configuration page to the user interface.

Furthermore, the properties of the intermediate model will have to be implemented. For a detailed discussion of those properties see Sect. 2.3. For this example, automatic `get` and `set` methods have been used (see listing A.2).

```
public List<IActivity> Activities { get; private set; }
public List<IArtifact> Artifacts { get; private set; }
public List<IRole> Roles { get; private set; }
public List<IDiscipline> Disciplines { get; private set; }
public List<IMilestone> Milestones { get; private set; }
public List<IAssociation> Associations { get; private set; }
public List<ITopic> Topics { get; private set; }
public List<ITask> Tasks { get; private set; }
```

Listing A.2: Implementation of intermediate model properties

Step 4 – (types). Before we are able to initialize the intermediate model, the types used have to be defined based on the interfaces provided by the PET core. For this tutorial, only the types *Milestone* and *Artifact* are implemented. Minimal implementations of the corresponding interfaces for `MyMilestone` and `MyArtifact` are shown in listing A.3.

```
class MyMilestone : IMilestone
{
    public string TypeName { get; set; }
    public DateTime ScheduledDate  { get; set; }
    public int ScheduleNumber { get; set; }
    public List<IMilestone> Predecessors { get; set; }

    public string Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public Dictionary<string, object> ExtendedData { get; private set; }

    public MyMilestone() {
        ExtendedData = new Dictionary<string, object>();
    }
}

class MyArtifact : IArtifact
{
    public bool IsInitial { get; set; }
    public bool IsExternal { get; set; }
    public bool HasDocumentTemplate { get; set; }
    public string DocumentTemplatePath { get; set; }
    public string DisciplineId { get; set; }
    public string TypeName { get; set; }

    public string Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public Dictionary<string, object> ExtendedData { get; private set; }

    public MyArtifact() {
        ExtendedData = new Dictionary<string, object>();
    }
}
```

Listing A.3: Implementation of intermediate model types

Step 5 – (method implementation). With the implementation of the two process elements used by the example process available, the initialization of the process model itself can begin. This will be done in the `Initialize` method.

Unlike a real-world process, the example process provider does not depend on external input. The example process will be fully defined in the `Initialize` method. To do so, the following steps are performed:

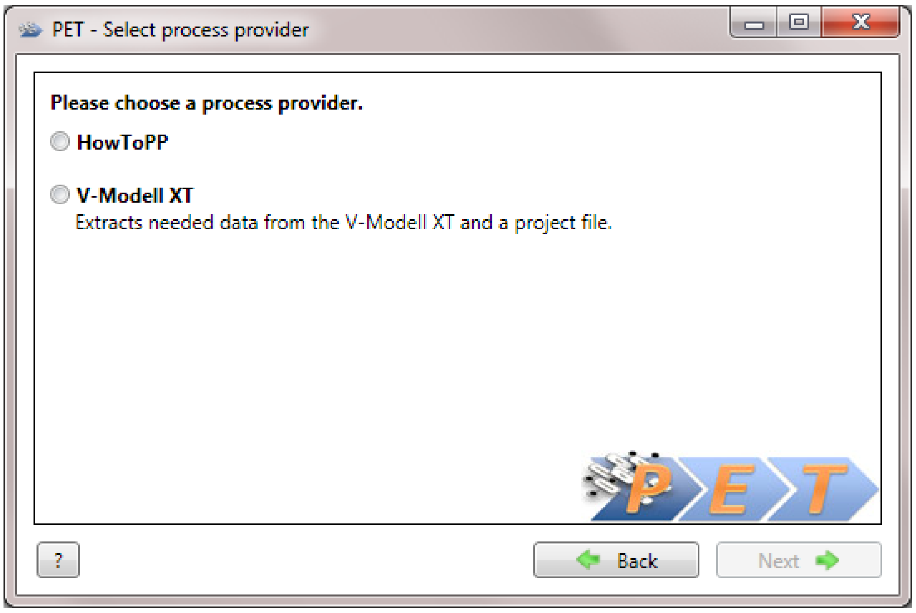
- Set a name for the process
- Initialize the intermediate model properties
- Create a milestone object *A Milestone* and add it to the intermediate model
- Create an artifact object *Specification for a pilot project* and add it to the intermediate model
- Create a dependency between the milestone and the artifact object

The source code for these steps can be found in listing A.4.

```
public void Initialize(Log log)
{
    Name = "HowTo␣process";

    Activities = new List<IActivity>();
    Artifacts = new List<IArtifact>();
    Roles = new List<IRole>();
    Disciplines = new List<IDiscipline>();
    Milestones = new List<IMilestone>();
    Associations = new List<IAssociation>();
    Topics = new List<ITopic>();
    Tasks = new List<ITask>();

    MyMilestone milestone = new MyMilestone();
    milestone.Name = "A␣milestone";
    milestone.TypeName = "Milestone";
    milestone.Id = "1";
    Milestones.Add(milestone);

    MyArtifact specification = new MyArtifact();
    specification.Name = "Specification␣for␣a␣pilot␣project";
    specification.TypeName = "Specification";
    specification.Id = "2";
    Artifacts.Add(specification);

    ArtifactToMilestone pmDependency = new ArtifactToMilestone(specification,milestone);
    Associations.Add(pmDependency);
}
```

Listing A.4: Initialization of the example process

The provider should now be ready to compile and run. If everything worked, the provider can be tested in PET – see Fig. A.4.


PET - Select process provider

Please choose a process provider.

HowToPP

V-Modell XT

Extracts needed data from the V-Modell XT and a project file.

Figure A.4.: First test of `MyProcessProvider`

A click on the *Next* button after selection of *HowToPP* will directly lead to the selection page for tool providers. This is because in this little example, the addition of a custom provider configuration page has been omitted. How such a page can be added is explained in the following tutorial on Tool Providers (see App. B).

B. How To: Implement a Tool Provider

The following tutorial will show the basic steps to create a simple tool provider for PET.

B.1. The Scenario

The small tool provider developed in this tutorial will gather all artifact types of the input process that the role *Project Leader* is responsible for or participates in their creation and output them to a text file.

B.2. Preparing the Tool Provider Project

The setup of the Visual Studio project for the tool provider is similar to the one for a process provider. Please see App. A.2 for detailed instructions.

Step 1 – (initial coding). Differing from the `IProcessProvider` interface, the interface `IToolProvider` has three methods to implement:

- `Initialize`: With this method, the tool provider receives a reference to the intermediate model in the form of `IProcessProvider`. Furthermore, a `Log` object is passed in that can be used to output messages to the user interface and to the log file.
- `GetConfigPageName`: This method is used to register the configuration page name with the framework to correctly include it in the user interface.
- `Process`: This method is called by the framework to perform the conversion from the intermediate model to the output format required by the target tool. In this scenario, only a text file will be written as described in the scenario (section B.1).

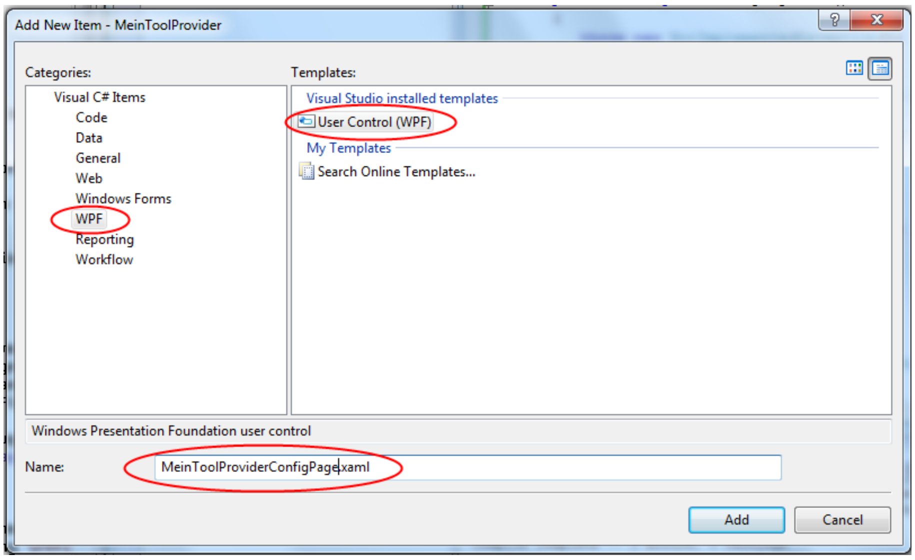
To realize the tool provider as described in the scenario, the references to the log object and to the intermediate model passed into `Initialize` are stored in instance members. The resulting source code is shown in listing B.1.

```
// ...
public class MyToolProvider : IToolProvider
{
        #region Private Attributes
        private IProcessProvider _processProvider;
        private Log _log;
        #endregion

        public void Initialize(IProcessProvider processProvider, Log log) {
                _processProvider = processProvider;
                _log = log;
        }
        // ...
```

Listing B.1: Implementation of Initialize

Step 2 – (adding a configuration page). Before the method `GetConfigPageName` can be implemented, the configuration page itself has to be added to the project. Unfortunately, the Visual Studio template *Windows Class Library* does not offer the WPF page object. The easiest way therefore is to add a WPF `UserControl` (see figure B.1) and change the resulting files by hand.

In the XAML file the `Tum.CollabXT` namespace is included and the base class of the control is changed to `CollabXT:ToolProviderConfigPage`. The modified XAML file is shown in listing B.2.


Add New Item - MeinToolProvider

Categories:
Visual C# Items
  Code
  Data
  General
  Web
  Windows Forms
  WPF
  Reporting
  Workflow

Templates:
Visual Studio installed templates
User Control (WPF)
My Templates
Search Online Templates...

Windows Presentation Foundation user control

Name: MeinToolProviderConfigPage.xaml

Figure B.1.: Add a WPF UserControl

```
<CollabXT:ToolProviderConfigPage x:Class="MeinToolProvider.MyToolProviderConfigPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:CollabXT="clr-namespace:Tum.CollabXT;assembly=Tum.CollabXT"
    Margin="10">
    <Grid>
      <!-- ... -->
   </Grid>
</CollabXT:ToolProviderConfigPage>
```

Listing B.2: XAML frame for the configuration page

Accordingly, the base class in the codebehind file has to be changed to `ToolProviderConfigPage`, too. Two methods of the base class have to be overloaded in the configuration page: `ReceiveWorkflowHandle` takes a reference to the flow control object. This can be used for example to manipulate the state of the Back and Next buttons. `ReceiveProviderHandle` takes a reference to the current process provider. Both references should be saved in instance members of the configuration page. An example listing is displayed in B.3.

```
using Tum.CollabXT;

namespace MeinToolProvider
{
   /// <summary>
   /// Interaction logic for MeinToolProviderConfigPage.xaml
   /// </summary>
   public partial class MyToolProviderConfigPage : ToolProviderConfigPage {
      #region Private Attributes
      private IConversionWorkflow _conversionWorkflow;
      private MyToolProvider _toolProvider;
      #endregion

      public MyToolProviderConfigPage() {
         InitializeComponent();
      }
      public override void ReceiveProviderHandle(IToolProvider toolProvider) {
         _toolProvider = toolProvider as MeinToolProvider;
      }
      public override void ReceiveWorkflowHandle(IConversionWorkflow workflow) {
         _conversionWorkflow = workflow;
      }
   }
}
```

Listing B.3: C# frame for the configuration page

Now the configuration page is ready to be registered with the PET framework. To do so, the method `GetConfigPageName` is changed as displayed in listing B.4.

```
// ...
public class MyToolProvider : IToolProvider {
        #region IToolProvider Members
        public string GetConfigPageName() {
            return "MeinToolProviderConfigPage.xaml";
        }
   // ...
```

Listing B.4: Implementation of `GetConfigPageName`

For the tool provider to show meaningful information on the user interface of PET, there are three more optional methods that can be implemented in the tool provider class.

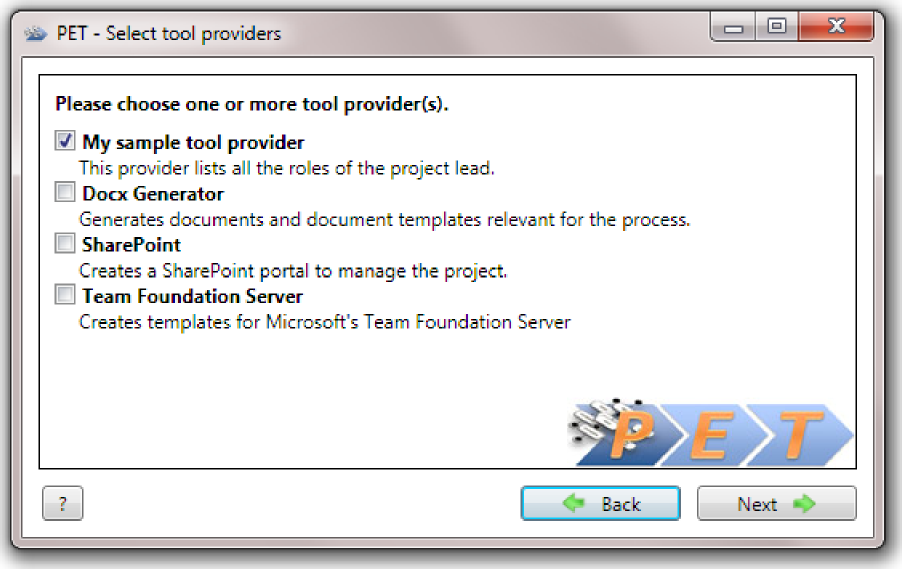
- `GetProviderName`: Here the name of the provider can be returned as it should appear in the PET wizard.
- `GetProviderDescription`: A more detailed description of the tool provider can be returned here. This will be displayed below the provider name on the provider selection page in PET.
- `GetProviderAuthor`: This can be used to return information about the provider developer. This information will be shown in the *About PET* dialog in the list of loaded plugins.

An example implementation of these three methods is shown in listing B.5.

```
// ...
public static string GetProviderName() {
        return "My␣sample␣tool␣provider";
}
public static string GetProviderDescription() {
        return "This␣provider␣lists␣all␣the␣roles␣of␣the␣project␣lead.";
}
public static string GetProviderAuthor() {
        return "...␣2010␣MyCompany";
}
// ...
```

Listing B.5: Informative output of the tool provider

The tool provider should now be ready for a first test. To do so, the project has to be compiled and the resulting assembly has to be copied to the PET directory if the output directory for the project has not already been changed. On the tool provider selection page in the tool there should now be an entry for the new tool provider as displayed in Fig. B.2.


PET - Select tool providers

Please choose one or more tool provider(s).

☑ **My sample tool provider**
This provider lists all the roles of the project lead.
☐ **Docx Generator**
Generates documents and document templates relevant for the process.
☐ **SharePoint**
Creates a SharePoint portal to manage the project.
☐ **Team Foundation Server**
Creates templates for Microsoft's Team Foundation Server

Figure B.2.: First Test of `MyToolProvider`

If the provider is selected and you click *Next* in PET, the empty configuration page should appear. The next task is to fill that page with life.

B.3. The Configuration Page

For the example scenario it is sufficient to offer the user a possibility to choose the output file. Four controls are added to the page to achieve this:

- A `Label` with a short description of what the user should do.
- Two `Label` for the file name itself. One for the name of the name of the field and one for the field content.
- A `Button` to browse for the file with a *Open File* dialog.

Step 1 – (control layout). The XAML code for those four controls is displayed in listing B.6.

```
<!-- ... -->

 <Grid>
   <Grid.RowDefinitions>
      <RowDefinition Height="32" />
      <RowDefinition Height="28" />
      <RowDefinition Height="*" />
   </Grid.RowDefinitions>
   <Grid.ColumnDefinitions>
      <ColumnDefinition Width="130" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="70" />
   </Grid.ColumnDefinitions>

   <Label Grid.ColumnSpan="3" Grid.Row="0" FontWeight="Bold" Content="Please␣enter␣
       the␣output␣file␣path." />

   <Label Grid.Row="1" Grid.Column="0" Name="lblOutputFile" FontWeight="Bold" Content
       ="Output␣file:" />
   <Label Height="28" Margin="0,0,5,0" Name="lblOutputFileValue" VerticalAlignment="
       Center" Grid.Row="1" Grid.Column="1" Foreground="Red"/>
   <Button Grid.Column="2" Margin="0,3,0,3" Grid.Row="1" VerticalAlignment="Center"
       Name="btnSetOutputFile" Click="btnSetOutputFile_Click" Content="..." />

</Grid>

<!-- ... -->
```

Listing B.6: Add Controls to the Configuration Page

Step 2 – (code behind). The listing B.6 already contains a click handler `btnSetOutput-File_Click` for the `Button`. This handler has to be implemented next in the codebehind file. Listing B.7 shows an example. Furthermore, a method method `UpdateControls()` has been added to activate or deactivate the *Next* button depending on the user input. It uses the method `Up-dateButtonState()` method from the flow control object that is of type `IConversionWorkflow`. This method accepts two boolean arguments: The first one controls the enabled state of the *Back* button. The second one controls the enabled state of the *Next* button. In the example (listing B.7), the *Back* button is always active, while the *Next* button only gets activated if the user has entered an output file.

```
public partial class MyToolProviderConfigPage : ToolProviderConfigPage
{
      #region Private Attributes
      private IConversionWorkflow _conversionWorkflow;
      private MyToolProvider _toolProvider;
      #endregion

      public MyToolProviderConfigPage() {InitializeComponent();}

      public override void ReceiveProviderHandle(IToolProvider toolProvider) {
         _toolProvider = toolProvider as MyToolProvider;
         UpdateControls();
      }
      public override void ReceiveWorkflowHandle(IConversionWorkflow workflow) {
         _conversionWorkflow = workflow;
         UpdateControls();
      }
      private void UpdateControls() {
         if (_toolProvider != null && !string.IsNullOrEmpty(_toolProvider.OutputFile)) {
            lblOutputFileValue.Content = _toolProvider.OutputFile;
            lblOutputFileValue.Foreground = new SolidColorBrush(Colors.Black);

            _conversionWorkflow.UpdateButtonState(true, true);
         }
         else {
            lblOutputFileValue.Content = "nicht␣gesetzt";
            lblOutputFileValue.Foreground = new SolidColorBrush(Colors.Red);

            _conversionWorkflow.UpdateButtonState(true, false);
         }
      }
      private void btnSetOutputFile_Click(object sender, EventArgs e) {
         SaveFileDialog sfd = new SaveFileDialog();
         sfd.Filter = "Textdateien␣(*.txt)|*.txt|Alle␣Dateien␣(*.*)|*.*";

         if (sfd.ShowDialog() ?? false) {
            _toolProvider.OutputFile = sfd.FileName;
            UpdateControls();
         }
      }
   }
```

Listing B.7: Code-Behind File of the Configuration Page

Step 3 – (additional properties). Listing B.7 also shows that the path to the output file is saved in a property of the tool provider class. That property has to be added to the `MyToolProvider` class accordingly (see listing B.8).

```
// ...
public class MyToolProvider : IToolProvider
{
   #region Private Attributes
   private IProcessProvider _processProvider;
   private Log _log;
   string _outputFile;
   #endregion

   #region Public Properties
   public string OutputFile
   {
      get { return _outputFile; }
      set { _outputFile = value; }
   }
   #endregion

   // ...
```

Listing B.8: `OutputFile` Property in `MyToolProvider`

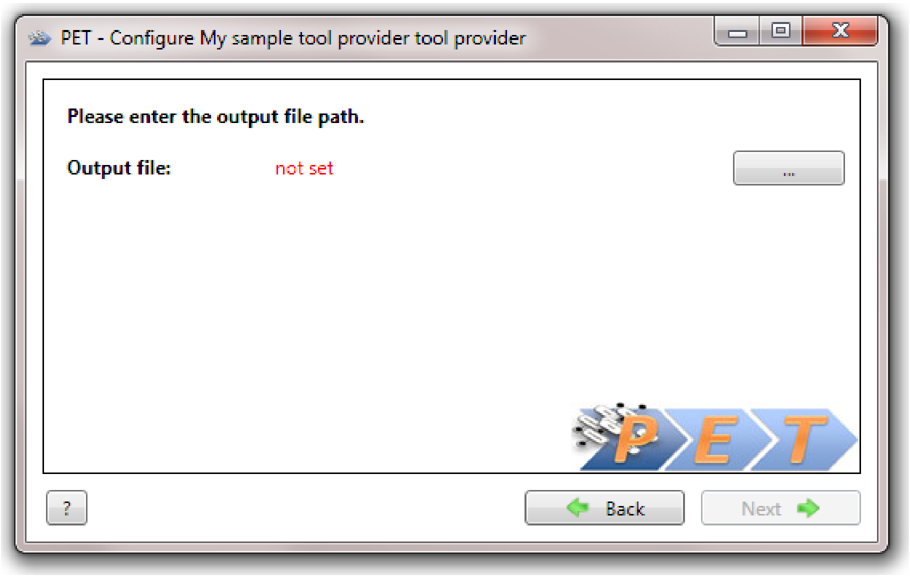
The configuration page of *My Example Tool Provider* should now look similar to Fig. B.3. The tool provider successfully hooks up with the PET framework, the graphical user interface works, the user can enter an output file and it's path is saved in a property of `MyToolProvider`.

What is left is to analyze the information about the development process provided by the process provider to extract the project leader role and the work products that he is responsible for or that he contributes to.

B.4. Interaction with the Intermediate Model

For the interpretation of the intermediate model and for the output to the target tool format, the `Process` in the tool provider has to be implemented accordingly. In this example, the target tool is just a text file. The necessary steps therefore are:

- Find the role *Project Leader*
- Find all artifacts associated with that role
- Write results to output text file


PET - Configure My sample tool provider tool provider

Please enter the output file path.

Output file: not set

Figure B.3.: Configuration Page of `MyToolProvider`

The listing B.9 shows the relevant part of the method `Process()` to find the *Project Leader* role.

```
IRole projectLeadRole = null;
// Find role project lead
_log.AddEntry("Searching␣role␣project␣lead␣...");
foreach (var role in _processProvider.Roles) {
        if (role.Name == "Project␣lead") {
                projectLeadRole = role;
                break;
        }
}
if (projectLeadRole == null) {
        _log.AddEntry("...␣not␣found", LogEntryType.Warning);
        return;
}
_log.AddEntry("...␣found!");
```

Listing B.9: Find the role project leader

With the help of the method `GetProductsByResponsibleRoleID()` and the method `GetProductsByParticipatingRoleID()` of the interface `IProcessProvider`, it is possible to obtain all artifacts associated with a role. The complete listing is displayed in listing B.10.

```
// Find all products associated to this role
_log.AddEntry("Find␣all␣associated␣products");
IArtifact[] responsibleProducts = _processProvider.GetArtifactsByResponsibleRole(
    projectLeadRole);
IArtifact[] participatingProducts = _processProvider.GetArtifactsByParticipatingRole(
    projectLeadRole);
```

Listing B.10: All artifacts associated with the project leader

The necessary data from the input development process is now there and ready to be written to the output file. The code doing that is shown in listing B.11.

```
_log.AddEntry("Find␣all␣associated␣products");
IArtifact[] responsibleProducts = _processProvider.GetArtifactsByResponsibleRole(
    projectLeadRole);
IArtifact[] participatingProducts = _processProvider.GetArtifactsByParticipatingRole(
    projectLeadRole);
// Write results to output file
_log.AddEntry("Writing␣products␣to␣output␣file");
using (TextWriter tw = new StreamWriter(_outputFile, false, System.Text.Encoding.Unicode
    )) {
    tw.WriteLine("#####################################################");
    tw.WriteLine("␣␣␣␣␣␣␣␣␣␣␣␣PRODUCTS␣OF␣PROJECT␣LEAD␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣");
    tw.WriteLine("#####################################################");
    tw.WriteLine();

    tw.WriteLine("The␣project␣lead␣is␣responsible␣for:");
    foreach (var product in responsibleProducts) {
        tw.WriteLine("\t" + product.Name);
    }
    tw.WriteLine();

    tw.WriteLine("The␣project␣lead␣is␣involved␣in:");
    foreach (var product in participatingProducts) {
        tw.WriteLine("\t" + product.Name);
    }

    tw.Close();
    _log.AddEntry("Done.");
}
```

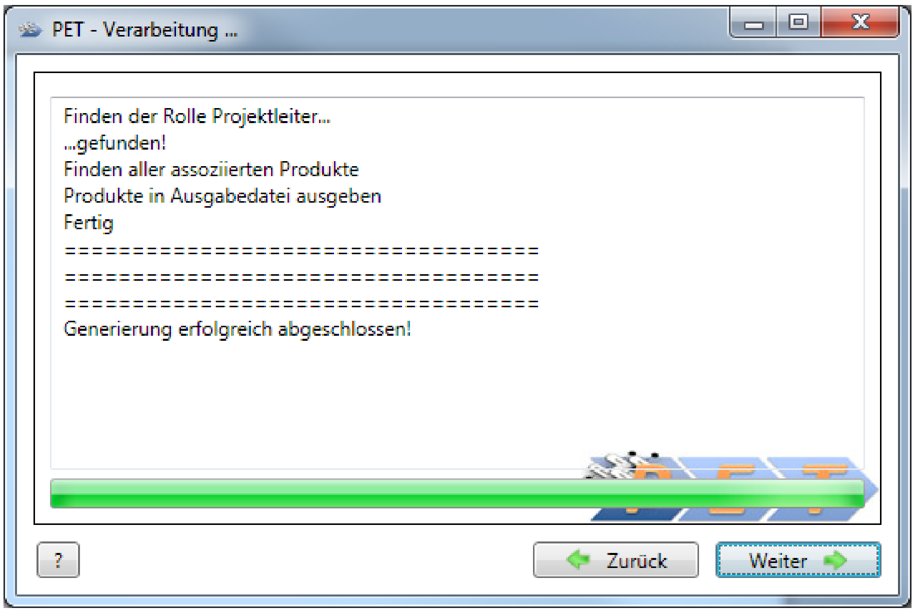
Listing B.11: Write results to output file

The result of applying `MyToolProvider` to a standard V-Modell XT (German variant) looks similar to the following:

```
#####################################################
        PRODUCTS OF PROJECT LEAD
#####################################################

The project lead is responsible for:
Arbeitsauftrag
Besprechungsdokument
Projektabschlussbericht
Projekthandbuch
Projektmanagement-Infrastruktur
Projektplan
Projektstatusbericht
Projekttagebuch
Risikoliste
Schätzung
Lieferung (von AN)
Projektabschlussbericht (von AN)
Projektstatusbericht (von AN)

The project lead is involved in:
Projektfortschrittsentscheidung
QS-Handbuch
Produktbibliothek
Anforderungen (Lastenheft)
Anforderungsbewertung
Abnahmeerklärung
Angebotsbewertung
Ausschreibung
Kriterienkatalog für die Angebotsbewertung
Vertrag
Vertragszusatz
```

B.5. Output of Log-Messages

The code examples B.9 and B.11 use the `Log` class to output information to the PET output window. The method `AddEntry()` takes as second argument a parameter of type `LogEntryType` that indicates the type of message. Warnings and errors will be highlighted with a color. An example of the output window is shown in Fig. B.4. In addition to the output in the output window, the messages are written to a log file (`log.txt` in the PET base directory). This can be especially helpful when looking for bugs in the provider.



```
Finden der Rolle Projektleiter...
...gefunden!
Finden aller assoziierten Produkte
Produkte in Ausgabedatei ausgeben
Fertig
=====================================
=====================================
=====================================
Generierung erfolgreich abgeschlossen!
```

Figure B.4.: Output of `MyToolProvider`

B.6. Persisting Settings in the PET Project File

To save and load settings in the PET project file, the methods `Serialize` and `Deserialize` have to be implemented. PET saves the settings in an XML file. For each provider, it automatically creates a XML node below which the provider can store it's individual settings. The example tool provider only has one parameter to persist: the path of the output file. The code to save that parameter can be found in listing B.12.

```
public void Serialize(System.Xml.XmlNode outputParentNode)
{
   XmlDocument xmlDoc = outputParentNode.OwnerDocument;

   XmlNode settingsNode = xmlDoc.CreateElement("settings");
   outputParentNode.AppendChild(settingsNode);

   XmlAttribute outputFileAttribute = xmlDoc.CreateAttribute("outputFile");
   outputFileAttribute.Value = _outputFile;
   settingsNode.Attributes.Append(outputFileAttribute);
}
```

Listing B.12: Save Settings

The corresponding code to load the output path is shown in listing B.13.

```
public void Deserialize(System.Xml.XmlNode inputParentNode)
{
    XmlNode settingsNode = inputParentNode.SelectSingleNode("settings");

    try
    {
    _outputFile = settingsNode.Attributes["outputFile"].Value;
    }
    catch (Exception)
    { }
}
```

Listing B.13: Load Settings

The example tool provider for PET is now complete and functional. Interaction with the intermediate model has been demonstrated, the configuration page has been hooked up with the PET user interface and the necessary interfaces have been implemented.

This small example together with the source code for more complex tool providers such as the one for SharePoint and the one for Team Foundation Server should give enough guidance for the development of a real-world tool provider.

Bibliography

[Bur02] M. Burghardt. *Einführung und Projektmanagement - Definition, Planung, Kontrolle, Abschluss*. Publics Corporate Publishing, 4 edition, 2002.

[FHKS09] Jan Friedrich, Ulrike Hammerschall, Marco Kuhrmann, and Marc Sihling. *Das V-Modell XT - Für Projektleiter und QS-Verantwortliche kompakt und übersichtlich*. Number ISBN: 978-3-540-76403-8 in Informatik im Fokus. Springer, 2. edition, oct 2009. available at http://www.springer.com/computer/programming/book/978-3-540-76403-8.

[KK03] P. Kroll and P. Kruchten. *The Rational Unified Process Made Easy – A Practinioner's Guide to RUP*. Addison-Wesley, 2003.

[KK08a] Marco Kuhrmann and Georg Kalus. Providing Integrated Development Processes for Distributed Development Environments. In *Workshop on Supporting Distributed Team Work at Computer Supported Cooperative Work (CSCW 2008)*, nov 2008.

[KK08b] Marco Kuhrmann and Georg Kalus. Werkzeugspezifisches Tailoring für das V-Modell XT. Forschungsbericht TUM-I0804, Technische Universität München, feb 2008.

[KKD08] Marco Kuhrmann, Georg Kalus, and Norbert Diernhofer. Generating Tool-based Process-Environments from formal Process Model Descriptions – Concepts, Experiences and Samples. In C. Pahl, editor, *Proceedings of the IASTED International Conference on Software Engineering (SE 2008) as part of the 26th IASTED International Multi-Conference on Applied Informatics*, number ISBN: 978-0-88986-715-4. ACTA Press, 2008. available at `http://www.actapress.com`.

[Kuh08a] M. Kuhrmann. *Konstruktion modularer Vorgehensmodelle*. PhD thesis, Technische Universität München, 2008.

[Kuh08b] Marco Kuhrmann. CollabXT: Kollaboration und verteilte Entwicklung mit dem V-Modell XT. *OBJEKTspektrum*, (März/April, Nr. 2):61–65, feb 2008. Schwerpunktheft: Globale Softwareentwicklung.

[Kuh08c] Marco Kuhrmann. Integration des V-ModellÆXT im Visual Studio Team Foundation Server – Erfahrungen aus dem Projekt CollabXT. In *1. Workshop: Integration von heterogenen Werkzeugen im agilen Zeitalter (IntegrA 08) im Rahmen der Software-Engineering-Konferenz 2008, München*, feb 2008.

[Mic07] Microsoft Corporation, editor. *Team Development with Visual Studio Team Foundation Server*. Number ISBN-13: 978-0735625716. Microsoft Press, 2007.

[TK09] T. Ternité and M. Kuhrmann. Das v-modell xt 1.3 metamodell. Forschungsbericht TUM-I0905, Technische Universität München, 2009.