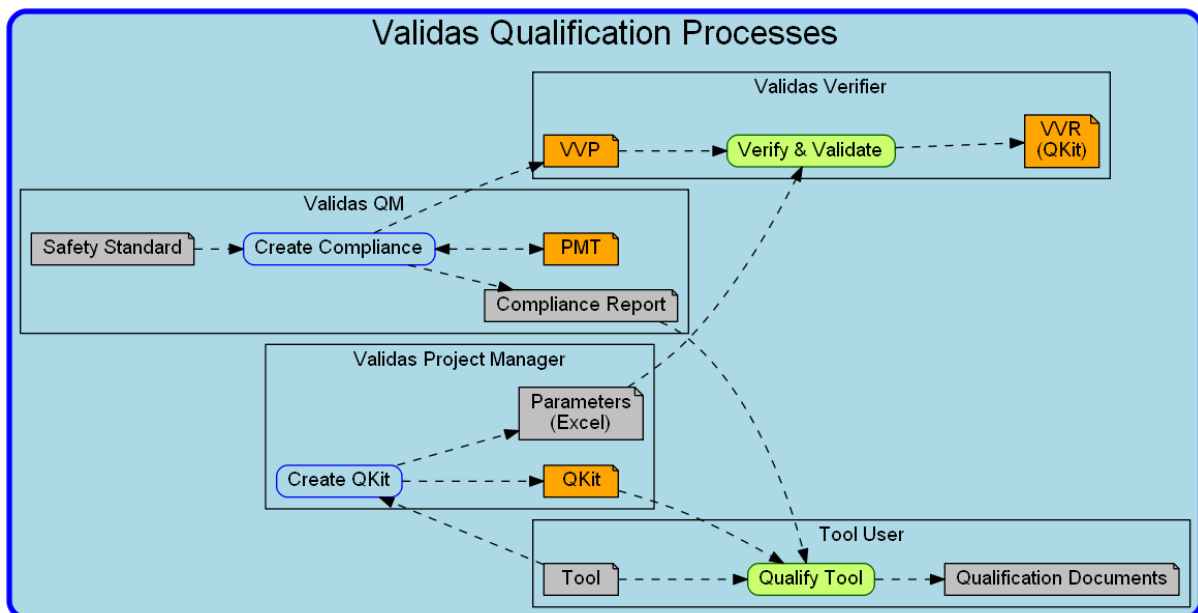


User Manual

The Validas

Process Modeling Tool¹

Version 1.2



Overview:

The Process Modeling Tool (PMT) is a process modeling tool for safety relevant processes. It supports the compliance argumentation with safety standards in the model and generates compliance reports as well as process reports and model-based verification and validation plans. Those are managed using the Verification and Validation Tool (VVT), which can read the .vvt files generated from PMT.

Based on the process model PMT can also be used to manage the process.

¹ This work has been developed within the German research project SPEDIT. SPEDIT was supported by the Bundesministerium für Bildung und Forschung (BMBF).

Contents

1	Introduction	5
2	Glossary	6
3	Safety Strategy	8
4	Compliance Method	10
5	Process Notation	13
5.1	General	14
5.2	Process View	15
5.3	Artifact View	16
5.4	Stake Holder View	17
6	Installation & Starting PMT	18
7	User Interface	22
7.1	Tool Menus	23
7.1.1	File Menu	23
7.1.2	Edit Menu	24
7.1.3	PMT Editor Menu	24
7.1.4	Window Menu	24
7.1.5	Help Menu.....	25
7.2	Tree Browser	25
7.2.1	On Various Elements	27
7.2.1.1	Show DOT (Graphviz)	27
7.2.1.2	Export DOT (Graphviz)	28
7.2.1.3	Infer and Set Types	28
7.2.1.4	Check Types	29
7.2.2	On Process Element	30
7.2.3	On ProcessModule Element	31
7.2.4	On Model Element.....	32
7.2.5	On Requirement Element	34
7.2.6	On Parameter, Binding, EnumValue and EnumType Elements .	37
7.3	Process Module View	38
7.4	Compliance View	39
7.5	Projection View.....	39
7.6	Diagnostic View	43
8	Features of PMT	44
8.1	Terms and Types	44
8.2	Process Modeling	49
8.2.1	Processes.....	49
8.2.2	Model-Based Processes.....	52
8.2.3	Requirements.....	54
8.2.4	Compliance	56
8.2.5	Reuse & Linking.....	59

8.2.6 Tailoring	61
8.2.7 Instantiation.....	62
8.2.8 Variables & Variants	66
8.2.9 Layouting.....	66
8.2.10 Tools	69
8.2.11 Project Management	70
8.2.11.1 Project Overview.....	71
8.2.11.2 Project Status Details	72
8.2.11.3 Excel-Interface for Project Management.....	73
8.2.11.4 Project Status Update	74
8.3 Consistency & Process Interfaces	75
8.4 Refinement	77
8.5 Validation	82
8.5.1 Syntactic Validation.....	83
8.5.2 Graphical Validation	85
8.5.3 Semantic Validation	85
8.6 Interfaces	85
8.6.1 ProcessModule Ex- and Import	85
8.6.2 VVT.....	86
8.6.3 Excel	88
8.6.3.1 Parameters Interface	88
8.6.3.2 Process Status	91
8.6.3.3 Process Description Export.....	92
8.6.3.4 Development Interface Agreement.....	93
8.6.3.5 Offer	94
8.6.4 Ecore Importer	96
8.7 Report Generators	97
8.7.1 Process Report	97
8.7.2 Compliance Report.....	98
8.8 Preferences.....	99
8.9 Filter Scoping	100
9 Meta Model of PMT	103
9.1 Syntax of Meta Model.....	104
9.2 Enumerations.....	106
9.2.1 ProcessStatus.....	106
9.2.2 SafetyLevel	107
9.2.3 Cardinality	107
9.3 General Interfaces	107
9.3.1 Named	108
9.3.2 Variantable -> Named	109
9.3.3 Verification Interface	110
9.4 Scoping, Hierarchy and Reuse	111
9.5 Types.....	112
9.5.1 Type -> Named	113
9.5.2 EnumType -> Type	114
9.5.3 EnumValue -> Named	114
9.5.4 ListType -> Type	115

9.6	Terms	116
9.6.1	Term	117
9.6.2	EnumValueRef -> Term	118
9.6.3	Constant -> Term	118
9.6.4	ListTerm -> Term	119
9.6.5	BoolTerm -> Term	119
9.6.6	ParamRef -> Term	120
9.6.7	InList -> BoolTerm.....	121
9.6.8	ORTerm -> BoolTerm	121
9.6.9	ANDTerm -> BoolTerm	122
9.6.10	NOTTerm -> BoolTerm.....	122
9.6.11	EQTerm -> BoolTerm.....	123
9.7	Bindings	123
9.8	Parameters	124
9.8.1	Parameter -> Named	125
9.8.2	ProcessParameter -> Parameter	126
9.8.3	PlanningParameter -> Parameter	126
9.8.4	ProjectParameter -> Parameter	126
9.8.5	ProcessVariable -> Parameter	127
9.9	Process Frame.....	127
9.9.1	Process -> Named	128
9.9.2	ToBeImplemented: Preference.....	130
9.9.3	History Records	130
9.10	Process Models	134
9.10.1	Process Module -> Variantable	135
9.10.2	Artifact -> Variantable	139
9.10.3	StakeHolder -> Variantable	141
9.10.4	VerificationModule -> ProcessModule	142
9.10.5	Criterion -> Variantable	144
9.11	Requirements & Compliance.....	144
9.11.1	Requirement -> Variantable	145
9.11.2	Compliance -> Variantable	146
9.12	Model-Based Processes.....	148
9.12.1	Model -> Artifact.....	148
9.12.2	MetaModel.....	149
9.12.3	MetaModelElement -> Named	150
9.12.4	MetaModelAttribute -> Named	150
9.12.5	MetaModelAssociation -> Named.....	151
9.13	Tools.....	151
9.13.1	Tool -> Variantable	152
9.13.2	Method -> Named	153
10	Known Issues.....	153
11	Licenses & Liability	154
12	Examples & Further Documentation	155

1 Introduction

The Process Modeling Tool (PMT) is a tool that supports model-based process modeling, safety standard compliance as well as the preparation of safety plans and safety cases via the interface to the verification and validation tool (VVT). See Section 3 for an overview how PMT fits into the safety strategy.

The PMT can be used for different purposes:

- Process Modeling, formalization and documentation with
 - Parameterized process
 - Automated tailoring
 - BPMN like notation of processes
 - Syntactic and semantic validation of process models
- Compliance with safety requirements and compliance argumentations ("safety plan") including GSN visualization
- Preparation of project specific verification and validation ("safety case") including interface to Verification and Validation Tool (VVT)
- Project Management Tool: planning and status management of projects (including process instances)
- Report generators for
 - Compliance reports
 - Process reports
 - Verification and Validation Report (see VVT)

This user manual describes the PMT tool with the following aspects:

- A glossary of the used terms in Section 2.
- The safety strategy in which PMT can be used in Section 3.
- The compliance Method, see Section 4.
- The used graphical process notation, see Section 5.
- The Installation, see Section 6.
- The user interface of PMT, see Section 7.
- The Features of PMT, see Section 8.
- The details of the model, see Section 9.
- The known bugs of PMT, see Section 10.
- Licenses are described in Section 11.
- References to examples & further documentation can be found in Section 12.

2 Glossary

The following abbreviations are used in the document.

- AOC: Anomalous Operating Condition
- Artifact: Element exchanged between processes
- BPMN: Business Process Model and Notation
- CR: Compliance Report²
- CT: Construction Task (during QKit creation)
- GSN: Goal Structuring Notation
- KB: Known Bug
- LCR: Library Classification Report
- LQP: Library Qualification Plan
- LQR: Library Qualification Report
- LSM: Library Safety Manual
- LTG: Library Test Generator
- PCCP: (Development) Process Compliance Check Plan
- PCCR: (Development) Process Compliance Check Report
- PMT: Process Modeling Tool
- Process Module: modular tasks in the process
- PT: Preparation Task (before QKit creation)
- Role: see Stakeholder
- QKit: Qualification Kit
- QP: Qualification Plan (general), can be LQP or TQP
- QR: Qualification Report (general), can be LQR or TQR
- QST: Qualification Support Tool
- SEOOO: Safety Element Out Of Context according to ISO 26262
- SM: Safety Manual (general), can be LSM or TSM
- Stakeholder: abstract person taking over responsibilities in the process
- SWC: Software Component, e.g. a library³
- TAU: Test Automation Unit
- TCA: Tool Chain Analyzer
- TD: Tool Detection (part of TCL computation according to ISO 26262)
- TCL: Tool Confidence Level (according to ISO 26262)
- TCR: Tool Classification Report
- TI: Tool Impact (part of TCL computation according to ISO 26262)
- TQL: Tool Qualification Level (according to DO-330)

² Do not confuse with Classification Reports LCR and TCR.

³ Note that libraries can be both changes and unchanged software components.

- TQP: Tool Qualification Plan
- TQR: Tool Qualification Report
- TSM: Tool Safety Manual
- V&V: Verification and Validation
- Verification Module: special form of Process module used to verify an artifact in the process
- VVP: Verification and Validation Plan
- VVR: Verification and Validation Report
- VVT: Verification and Validation Tool
- VT: Verification task (after QKit creation)

3 Safety Strategy

PMT is the first step towards safe and efficient development processes. Safety is planned using PMT, by applying the compliance method, see Section 4. Efficiency is achieved using automatization with a safe tool chain and safe libraries. Safety of tools is achieved by classifying the tools (using the Tool Chain Analyzer) and by qualifying critical tools using qualification Kits. Safety of libraries is achieved by qualifying the used functions using library qualification kits. More information about the other qualification tools (TCA, QKits, VVT) can be found in <http://www.validas.de/en/tools/>.

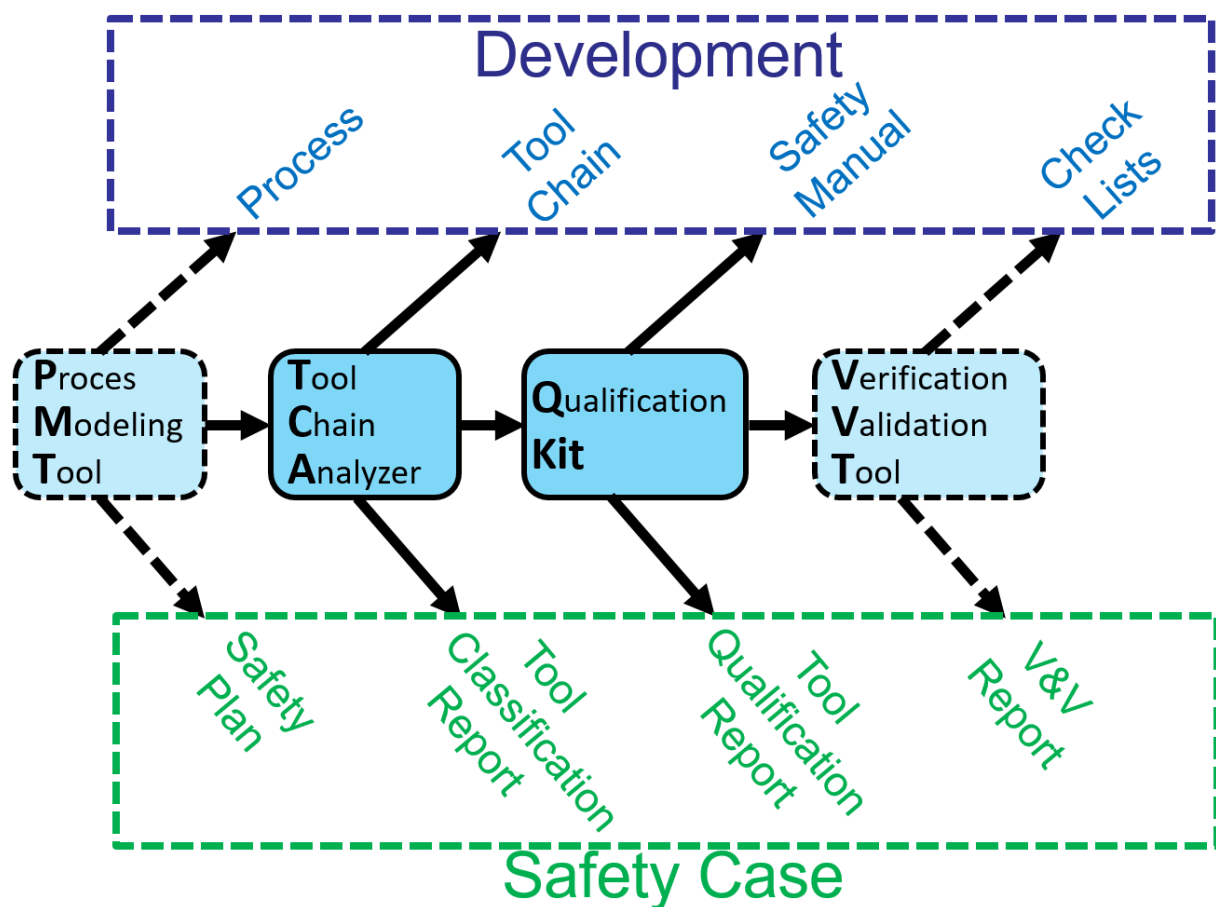


Figure 1: Safety Processes and Support Tools

Figure 1 shows how the Validas Tools build the interface between a safe development process and the safety case (TCA and QKit being the commercial tools, and PMT, VVT the free utilities that complete the tool chain):

- 1) PMT is used to plan the process and to show its compliance with the safety standard. The generated process description can be used

within development process, e.g. as reference, while the generated compliance report is the main part of the safety plan. Also the check list templates for Verification and Validation are generated from PMT and will be used later from VVT to ensure safety of the project.

- 2) Once the process is described in PMT the corresponding tool chain can be developed and modeled using the tool chain analyzer. In a future version TCA will be able to automatically create a model from the process model (that already contains a coarse tool model), e.g. by creating a use case in the tool chain for every process step that is linked to a tool in PMT. TCA can be used to document and analyze the tool chain and to make it safe by classifying the tools and providing safety manuals for the uncritical tools, i.e. those tools that can be used sufficiently carefully without having manual extra work. Also other tools can be classified as uncritical, e.g. by enriching the process with extra activities.
- 3) The critical tools, i.e. the tools that the user wants to rely on the function without checking its outputs can be qualified using so called qualification kits. QKits typically base on validation, i.e. testing the tool and can for example be easily build using the Validas framework. QKits contribute to the safety case with all required documents, mainly the tool qualification report and generate a tailored safety manual covering exactly the use cases of the tools.
- 4) At the end of the development process (or after some development steps) the product has to be verified and validated. This is done using the Verification and Validation Tool. It provides the check lists to the project, manages their results and generates the Verification and Validation Report to complete the safety case.

Since classification (TCA) and qualification (QKit) is the core business of Validas and since PMT is the entry ticket to safe development (including classification and qualification), Validas decided to make it freely available to the world (including VVT).

4 Compliance Method

The applied compliance method is model based. It bases on a parameterized model that is used to model the following things:

- Requirements from safety standard (or from somewhere else),
- Development process of the qualification kit (or something else),
- Compliance argumentation,
- Verification actions and
- Parameters.

The key ideas of the compliance method are:

- 1) Every requirement is linked to two things:
 - a. an element in the process that implements it and
 - b. a verification step in the process that verifies the corresponding artifact
- 2) Project parameters: Every project is different, even if it follows the same process. Those differences are modeled using parameters in the process. The parameters are instantiated with values during the project. In our qualification projects typical parameters are "TEST" or "FEATURE".

Parameters can be used for project management, e.g. qualifying 20 features and creating 100 tests, but parameters have to be used for verification and validation in order to ensure requirement (and standard) compliance.

The compliance method is therefore structured in two parts

- 1) The process specific part, that is the scheme for all projects.
- 2) The project specific part, which is just an instantiation of the process by defining the parameters and performing V&V for all instances as pre-scribed within the process.

The process (part 1 only) can be assessed independently. For example Validas has a TÜV certification for the processes of Tool qualification (since 2018), based on this compliance method.

This compliance documentation is, together with the process report, the basis for a process certification for the process.

The compliance method is graphically shown in Figure 2. The compliance is achieved in the following steps:

- 1) Process Part. We use the Process Modeling Tool (PMT) for this:
 - a. Model or select requirements for the process.
 - b. Model or configure the process (based on existing processes), including the verification steps.

- c. Argue the compliance of the process by providing for every requirement at least one implementing process and one verification action.
 - d. Generate the process report (PR).
 - e. Generate this compliance report.
 - f. Generate the verification and validation plan.
- 2) Project specific part (only the verification and validation, which is essential for the compliance). We use the V&V Tool VVT for this
- a. Instantiate the Verification and Validation Plan (VVP) by
 - i. Assigning concrete names to stakeholders that perform V&V
 - ii. Import the project parameters. For qualification projects based on Validas Tool Chain Analyzer this can be done by exporting the parameters from the TCA tool into an Excel table)
 - b. Perform V&V by going through all checks for all instances of the parameters (this can be done using Excel Export & import of VVT)
 - c. Check Completeness and generate the Verification and Validation Report (VVR) using VVT.

The safety plan is the description of the process including the compliance argumentation and the safety case is the safety plan including the V&V report with the verification results.

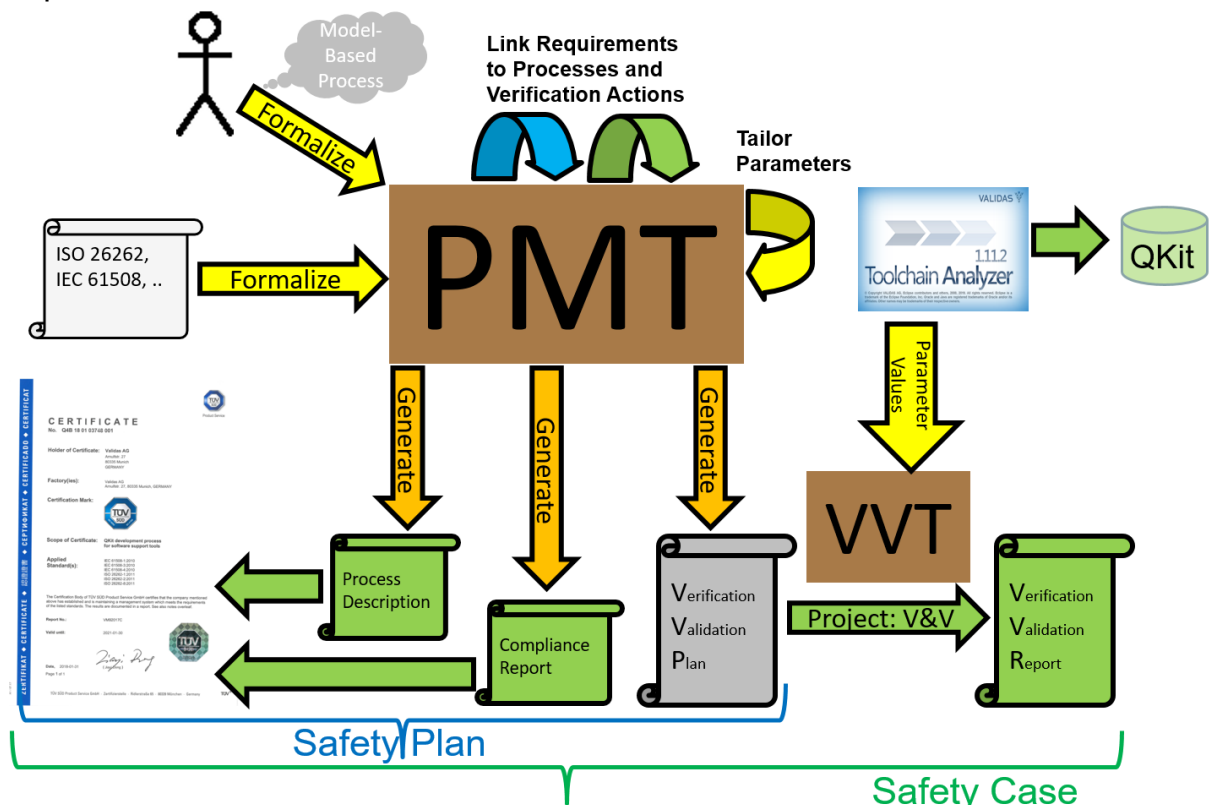


Figure 2: Compliance Method

As notation for the compliance argumentation we use a GSN-like (Goal Structuring Notation) notation with the following elements (see Figure 3).

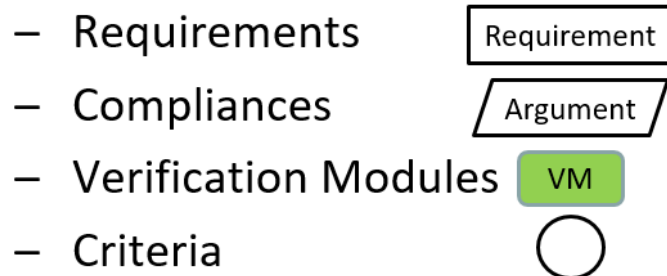


Figure 3: Used Goal Structuring Notation (GSN) Subset

For example an argumentation (“Compliance”) to meet a code coverage requirement by good test cases verified in two steps with four criteria could look like depicted in Figure 4.

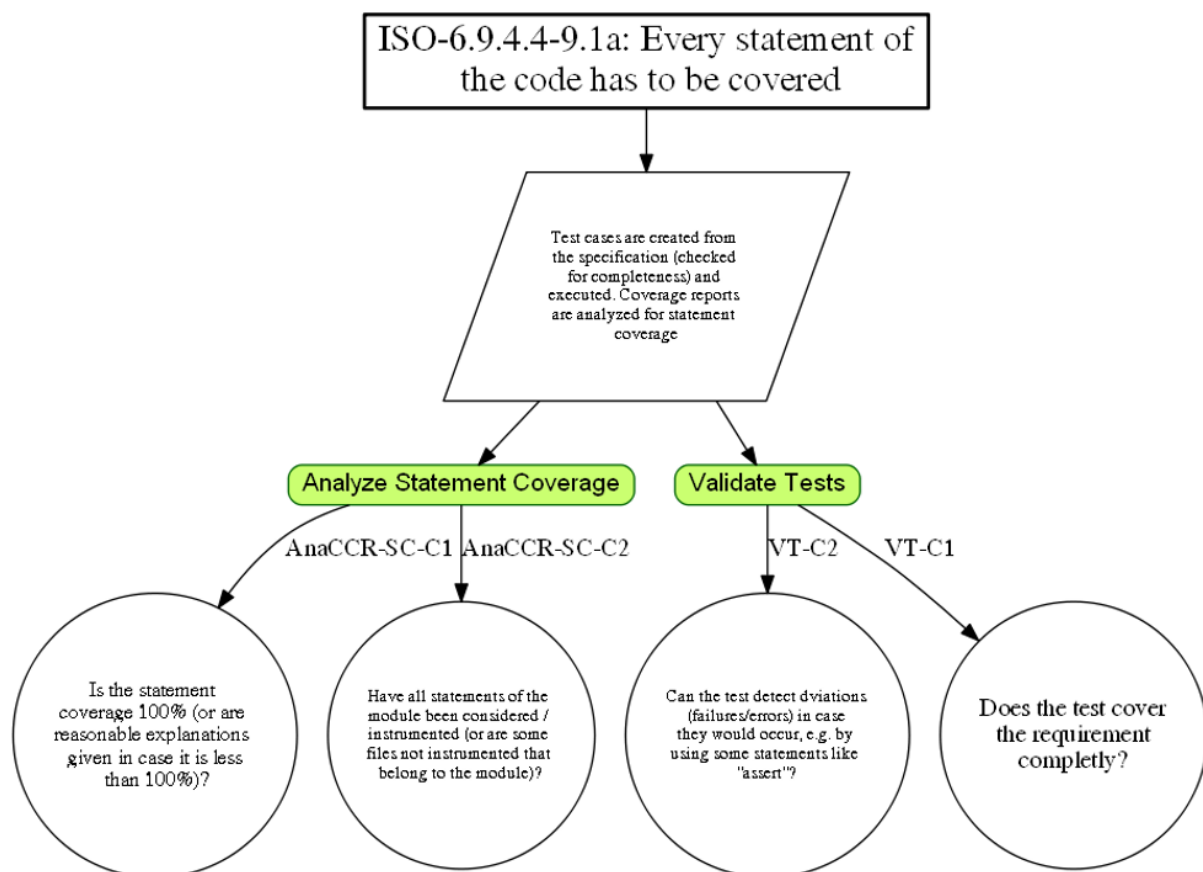


Figure 4: Example Compliance Argumentation using GSN

5 Process Notation

The used process notation is based on a model and consists of the following elements

- StakeHolder: A person responsible for Processes and Artifacts
- ProcessModule: A modular task/process with input & output artifacts
- Verification Module: a special module that verifies an artifact and checks that a requirement is satisfied by asking some review questions (called "Criteria")
- Model: a special form of an Artifact, allowing to define which modeling elements (e.g. Tool, Feature,.. in TCA Models) are mandatory (e.g. Tool-Name) and which are optional (e.g. Feature-Comment).
- Parameter: A parameter of the process indicating that the process has to be iterated for all values of the parameter. Parameters are used to tailor the process and to instantiate it. Parameters can be bound to values or list of values using Bindings. There are the following parameters:
 - Process Parameter: Describes a tailoring parameter, typically instantiated with concrete values. All elements that have variant terms with process parameters evaluating to false are removed.⁴
 - Project Parameter: A Parameter indicating instances of the process, typically instantiated with lists of values indicating that the process modules have to be repeated for each value of the parameter
 - Process Variable: Describing a condition in the process with alternative following processes. Process variables are not instantiated but determined when executing the process, e.g. "<REVIEW_OK> -> YES -> Release"

All elements are contained in a container called Process.

⁴ This process has already been tailored, such that all process variables have values and all unused process elements are removed.

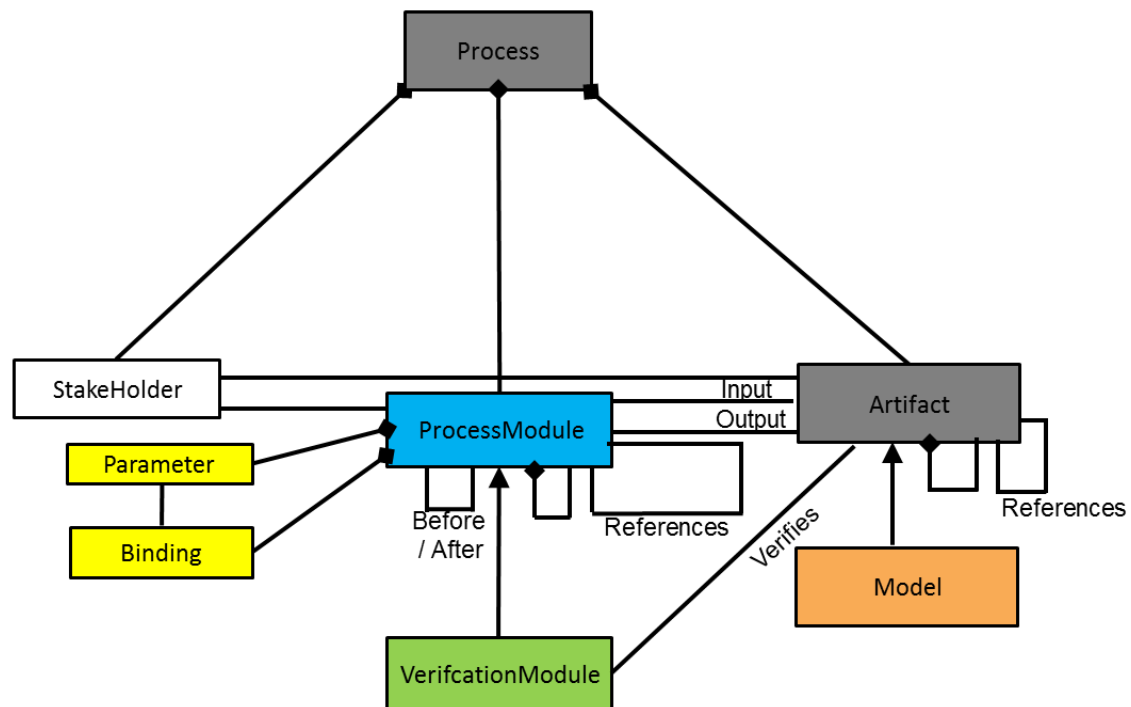


Figure 5: Main Process Description Elements and Relations

The graphical images are using a Business Process Modeling Notation BPMN like notation with “swim-boxes” instead of “swim-lines” to improve graphical layout.

The following graphical views are supported:

- Process View: describes the process modules, see Section 5.1.
- Artifact view: describes the structure of artifacts and their use, see Section 5.3.
- Role View: describes the roles with their responsibilities, see Section 5.4.

5.1 General

The graphical notation visualizes the following elements:




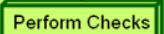



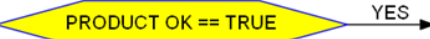
- ▶ **Process Module:** blue, rounded box: 
- ▶ **Verification Module:** green, rounded box: 
- ▶ **Hierarchical Process Module:** blue folders: 
- ▶ **Hierarchical Verification Module:** green folders: 
- ▶ **Artifact:** Grey box with note: 
- ▶ **Model:** Orange box with note: 
- ▶ **StakeHolder:** transparent box: 
- ▶ **Conditions:** yellow routes: 

Figure 6: Visualization of Process Model Elements

The graphical notation visualizes the following relations between elements:

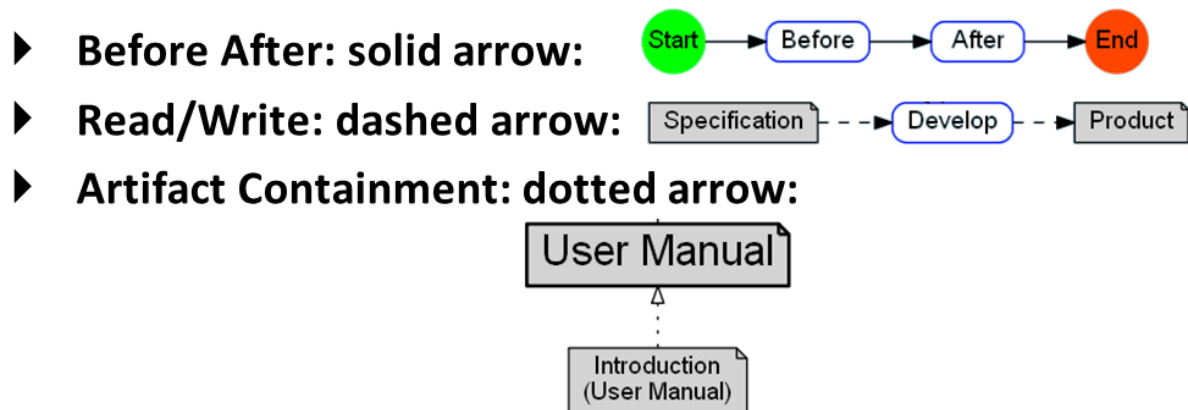


Figure 7: Relations between Model Elements

5.2 Process View

The process view describes a Process Module / Verification Module using the graphical notation of Section 5.1. The name of the selected module is written in double size. An example can be found in Figure 8.

It shows the process module “Develop Safe Product” and its sub-processes grouped into the swim-boxes of the involved stake holders.

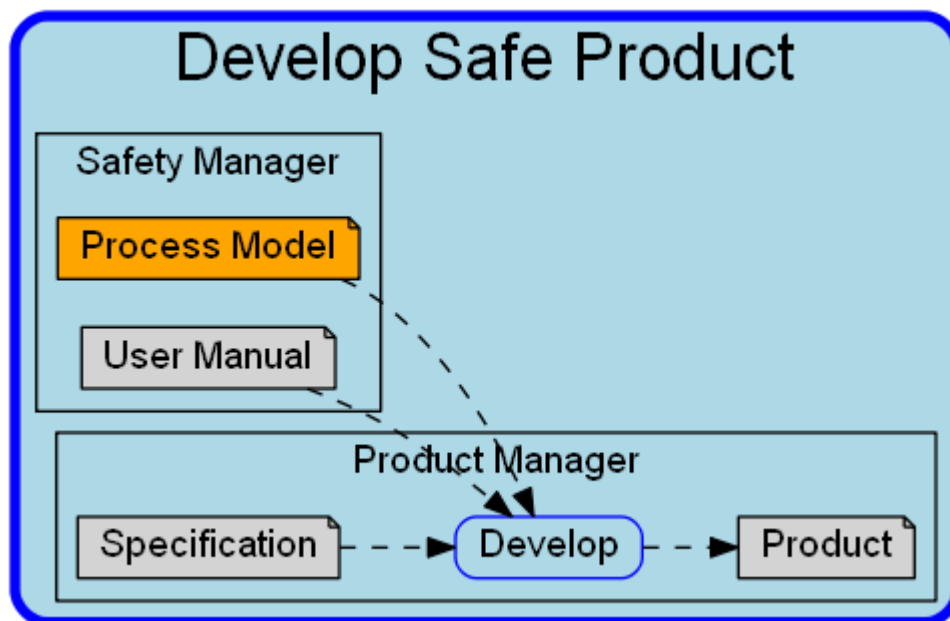


Figure 8: Process View for “Develop Safe Product”

In addition to this input output related modeling style, processes can also be modeled without artifacts using a sequential style by using the after/before relation as shown in the example in Figure 9. This shows a

simple process “Sell Product” as interaction between a sales manager and a customer.

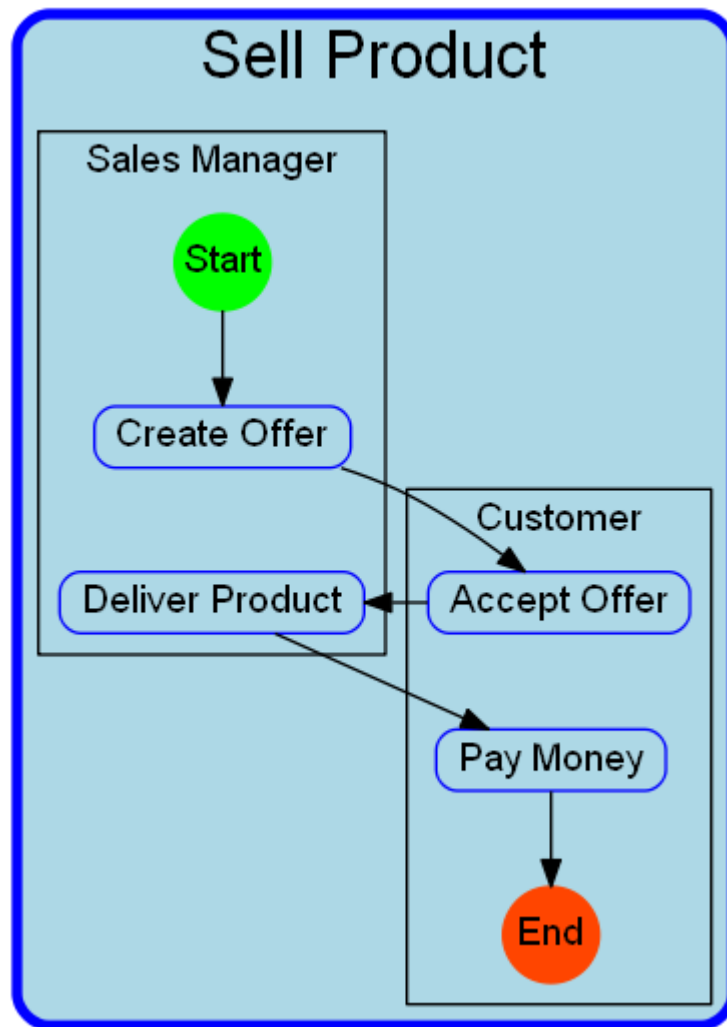


Figure 9: Sequential Process Module View for “Sell Product”

5.3 Artifact View

The artifact view describes an Artifact or Model using the graphical notation of Section 5.1. The name of the selected Artifact or Model is written in double size. An example can be found in Figure 10.

It shows the user manual and its content (here only the introduction) and that it is used from the product manager for development.

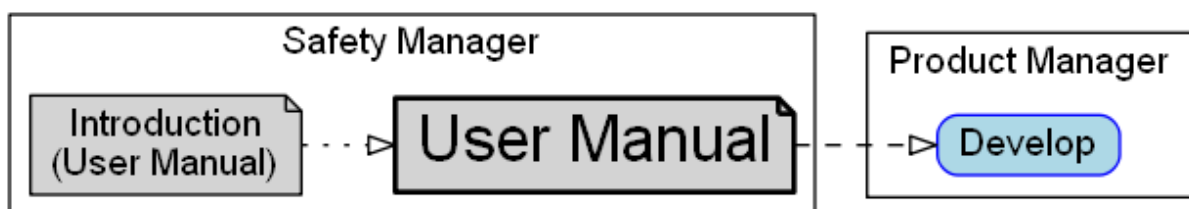


Figure 10: Artifact View for “User Manual”

5.4 Stake Holder View

The stake holder view describes a Stakeholder using the graphical notation of Section 5.1. The name of the selected Stakeholder is written in double size. An example can be found in Figure 11. It shows the two hierarchic processes owned by the Product Manager to produce a product and his responsibilities for the specification.

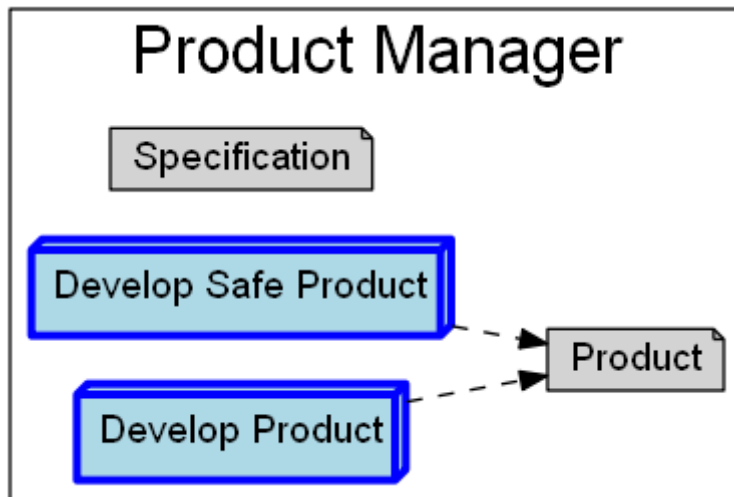


Figure 11: Stakeholder View for "Product Manager"

6 Installation & Starting PMT

To install the tool just unzip the distribution into a program folder in which you have **write permissions** and ensure that the PMT can create a workspace directory there, i.e. that there are no root privileges required to create the workspace there.

Furthermore, **Java 1.8 (JRE)** has to be installed on your computer and in the path, since new versions of PMT do not contain jre any more due to legal restrictions.

The ZIP file contains the PMT application: "ProcessModelingTool.exe" (see Figure 12).

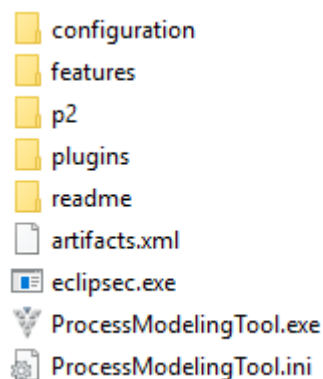


Figure 12: Distribution of Process Modeling Tool

In the plugins directory of the distribution, there are folders "de.validas.spm.pmt.documentation" containing this documentation. Furthermore, there is a plugin "de.validas.spm.pmt.examples" with some example models.

You can check the version of PMT in the About-Box see Figure 14 that can be started in the Help-Menu as shown in Figure 13

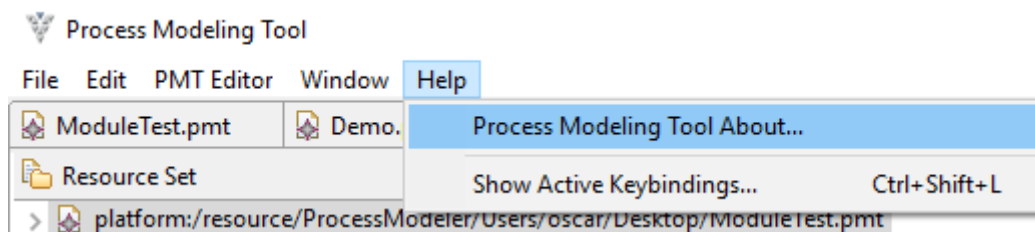


Figure 13: Starting About Box

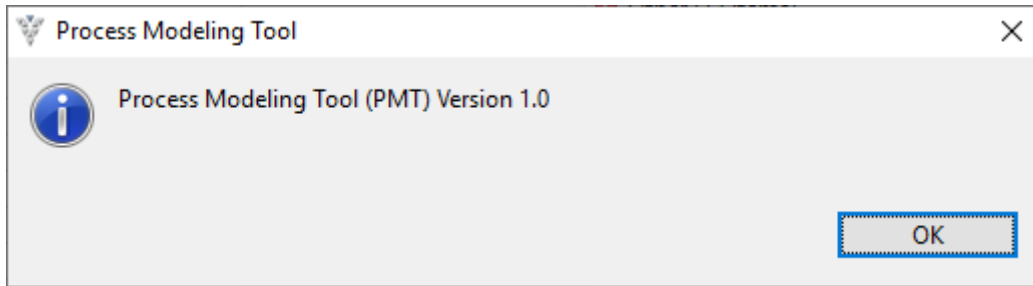


Figure 14: About Box with PMT Version

Note we document here only the PMT specific features and not general Eclipse mechanisms like the "Show Active Key Bindings" as visible in the Figure 13.

Requirements: If you want to include the automatically generated images in the report you have to install GraphViz. GraphViz is open source graph visualization software which is used by the PMT to generate the images. You can find the software and the installation manual at <http://www.graphviz.org>.

Note: GraphViz (dot.exe) has to be added into the search path of your system such that PMT can find it.

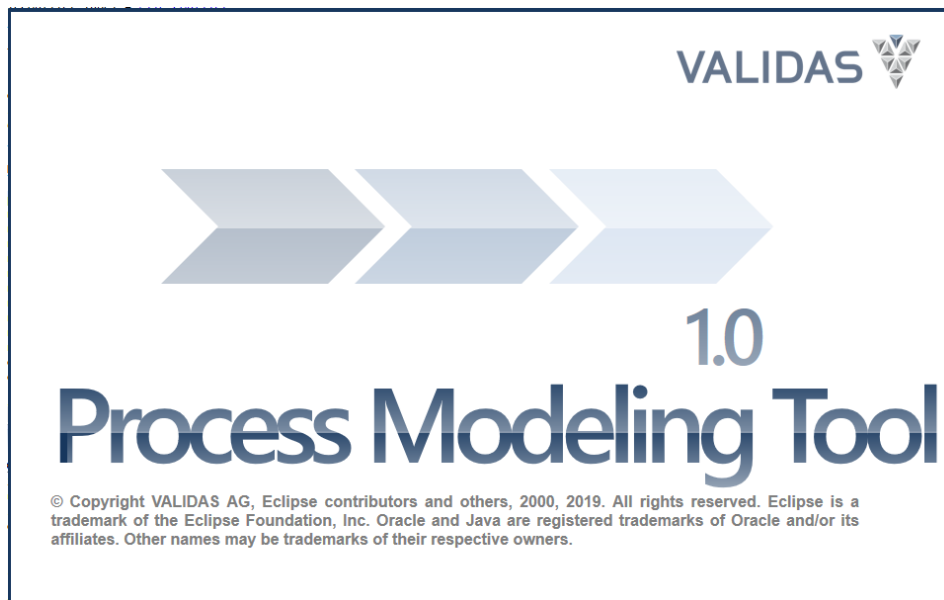


Figure 15: Splash during starting up of the Process Modeling Tool

After some seconds, the empty PMT starts (see Figure 15) and the user interface is ready for modeling. Initially it will be empty as depicted in Figure 16.

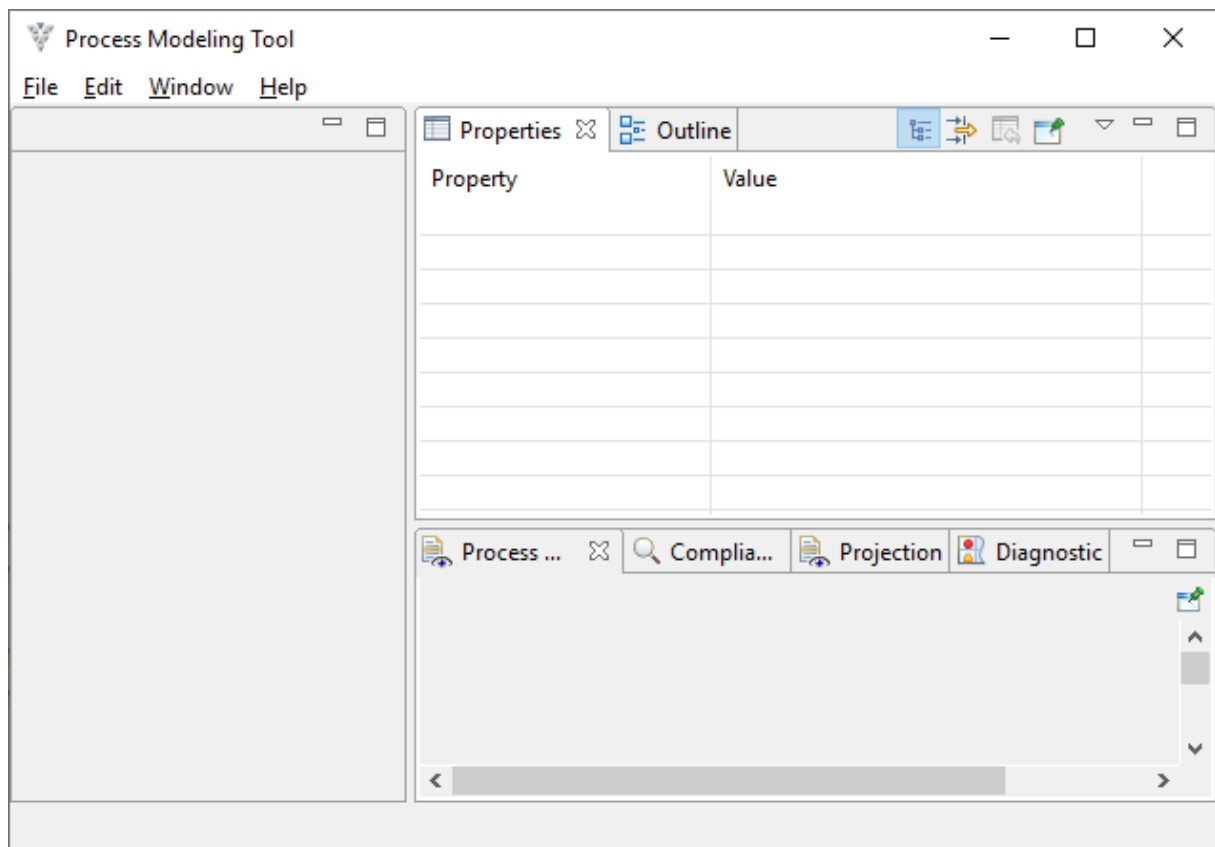


Figure 16: Empty PMT

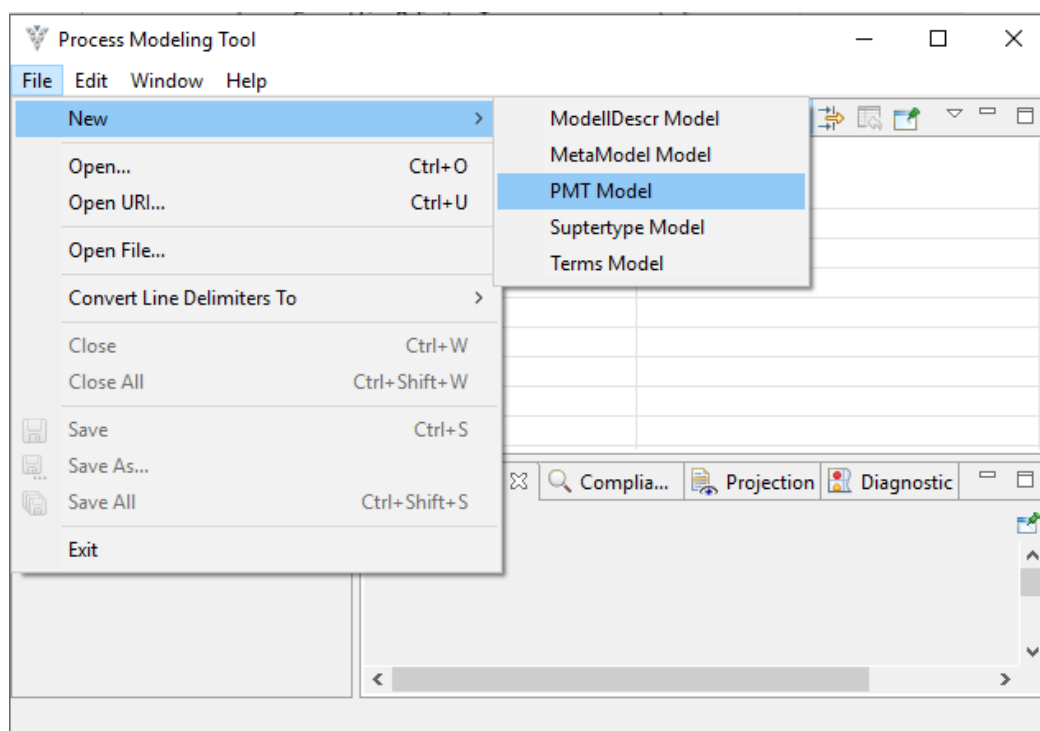


Figure 17: Create new PMT Model

To create new PMT models use the File -> New -> PMT Model dialog, see Figure 17 and select "Process" as main element, see Figure 18.

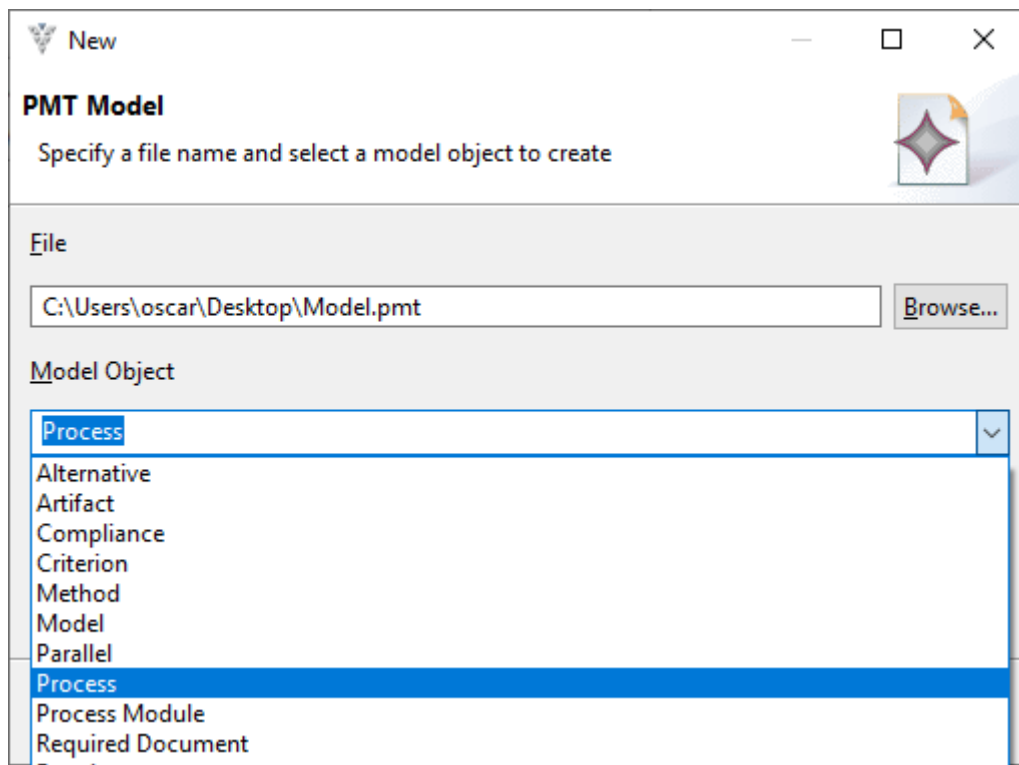


Figure 18:Select File and Model Object: Process

7 User Interface

The PMT user interface provides some standard menus. Main functionality is performed by actions that are directly started in the model tree browser.

PMT models are organized in a tree structure (see Section 9) but can be displayed with many different views. Some general actions (like opening views or setting preferences) are started from the menus in PMT, most actions are “popup actions” and are started directly from the tree browser.

Therefore, the user interface of PMT has the following components:

- Tool Menus, see Section 7.1,
- Tree Browser, see Section 7.2,
- Property Editor View, see Section 7.2.4,
- (Graphical) Process Module View, see Section 7.3,
- (Graphical) Compliance View, see Section 7.4,
- Projection View, see Section 7.5,
- Diagnostic View, see Section 7.6.

Figure 19 shows the views of the PMT. The tree browser is located on the upper left part, the property editor view on the right side. The other views are located at the right lower corner. All views can be modified and resized using the typical Eclipse mechanisms.

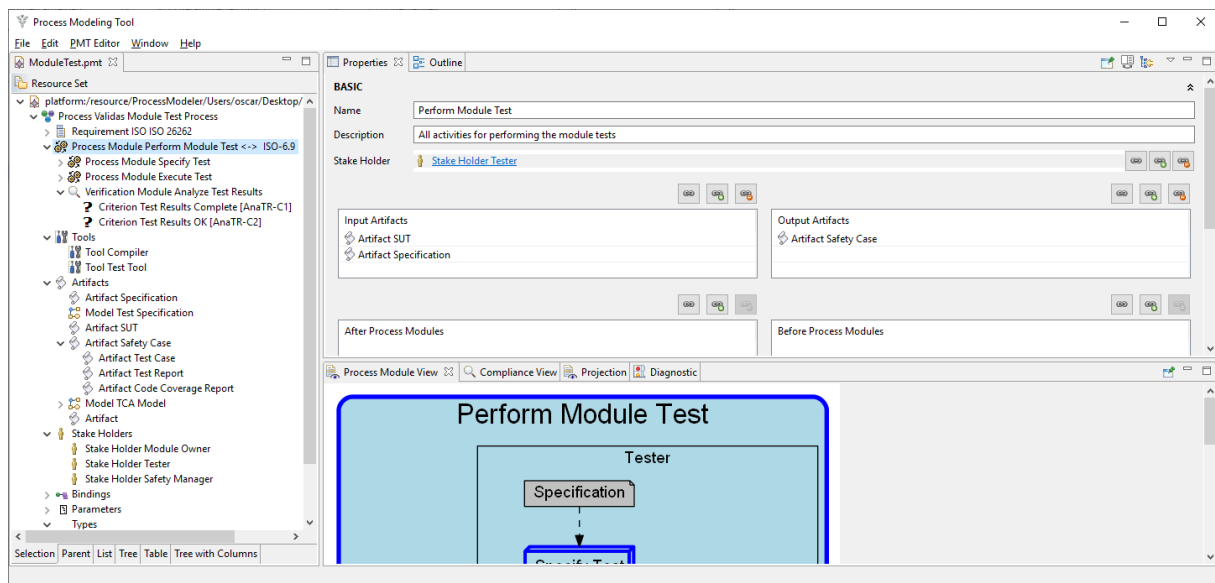


Figure 19: Views of PMT

7.1 Tool Menus

PMT offers the following tool menus (see Figure 20):

- File Menu, see Section 7.1.1,
- Edit Menu, see Section 7.1.2,
- PMT Editor Menu, see Section 7.1.3,
- Window Menu, see Section 7.1.4,
- Help Menu, see Section 7.1.5,

This section describes the offered functions shortly (most functions are default Eclipse modeling functions that are from the EMF).

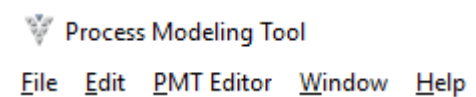


Figure 20: PMT Menus

7.1.1 File Menu

The file menu allows to open & close files and to create new files with PMT models. These are basic tool functions and work as expected.

Only important aspect is here, that for creation of a new model the function New -> PMT Model has to be chosen and the Root element "Process" should be selected, as described in Section 6.

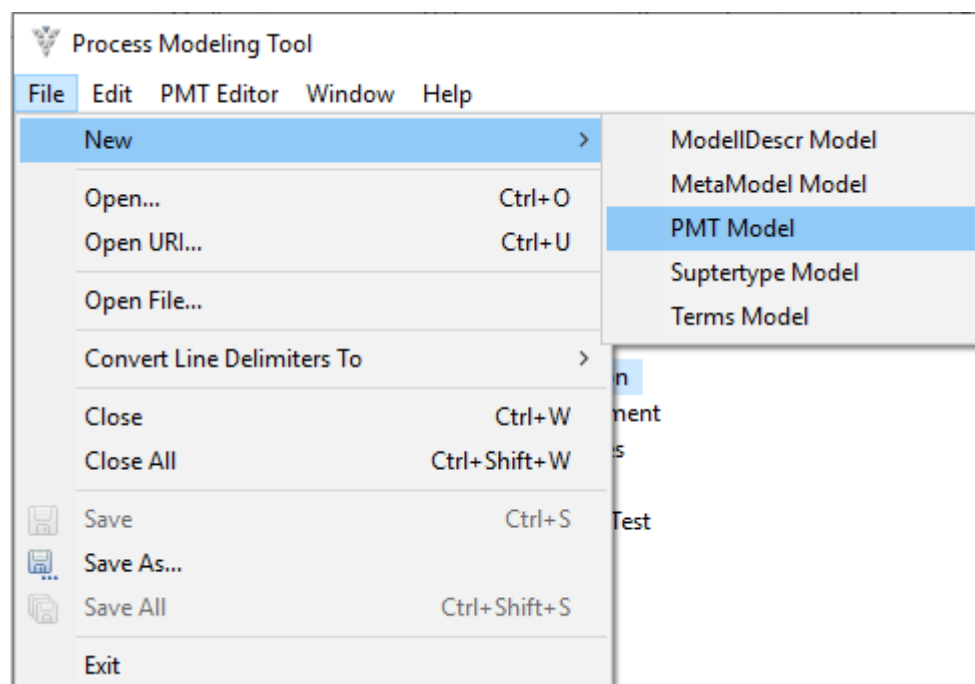


Figure 21: File Menu of PMT

7.1.2 Edit Menu

The edit menu allows to do simple editing commands (see Figure 22). All of them are available using standard shortcuts.

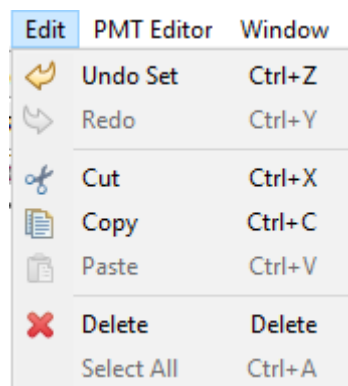


Figure 22: Edit Menu

7.1.3 PMT Editor Menu

The PMT Editor menu offers PMT Editing features, see Figure 23. The features are context dependent and a subset of the popup action menus. Their functionality is described in Section 7.2.

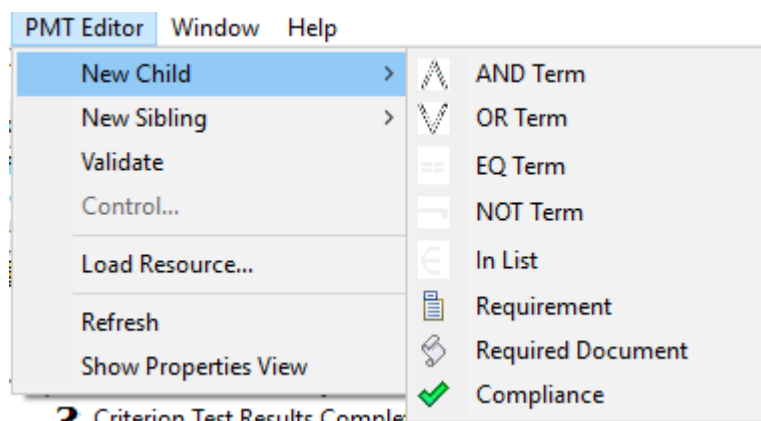


Figure 23: Tools Menu

However, since those functions are also available in the tree browser this menu is likely to be removed in future versions of PMT.

7.1.4 Window Menu

The window menu (see Figure 24) offers some useful functions regarding the window handling of TCA:

- 1) **Open in New Window**: opens PMT in a new window

2) **Preferences**: opens the preference dialogs, see Section 8.8.

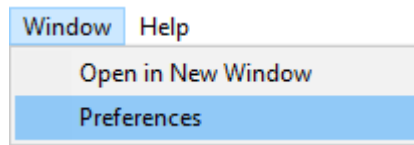


Figure 24: Window Menu

7.1.5 Help Menu

The help menu (see Figure 25) shows the about box of PMT and the active key bindings that might help non-Eclipse experts to get familiar with generic Eclipse features.

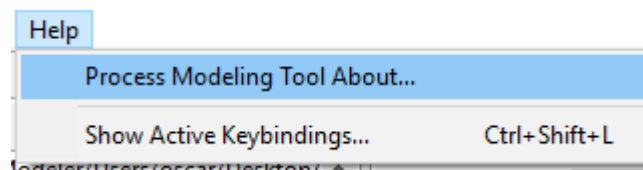


Figure 25: Help Menu

Further helpful information (that should be accessible here in future version of PMT) are:

- The PMT Examples in the examples plugin folder,
- This user guide, available in the documentation plugin.

7.2 Tree Browser

The tree browser allows to browse and edit the model structure and to start some actions on the model element with the right mouse button. The available modeling elements are described in Section 9, the actions in the following sub-sections.

Note there are several actions inherited from Eclipse that are currently useless in PMT and might be removed or implemented in future versions of PMT, e.g. "Team ->", "Compare With->", "Replace With ->".

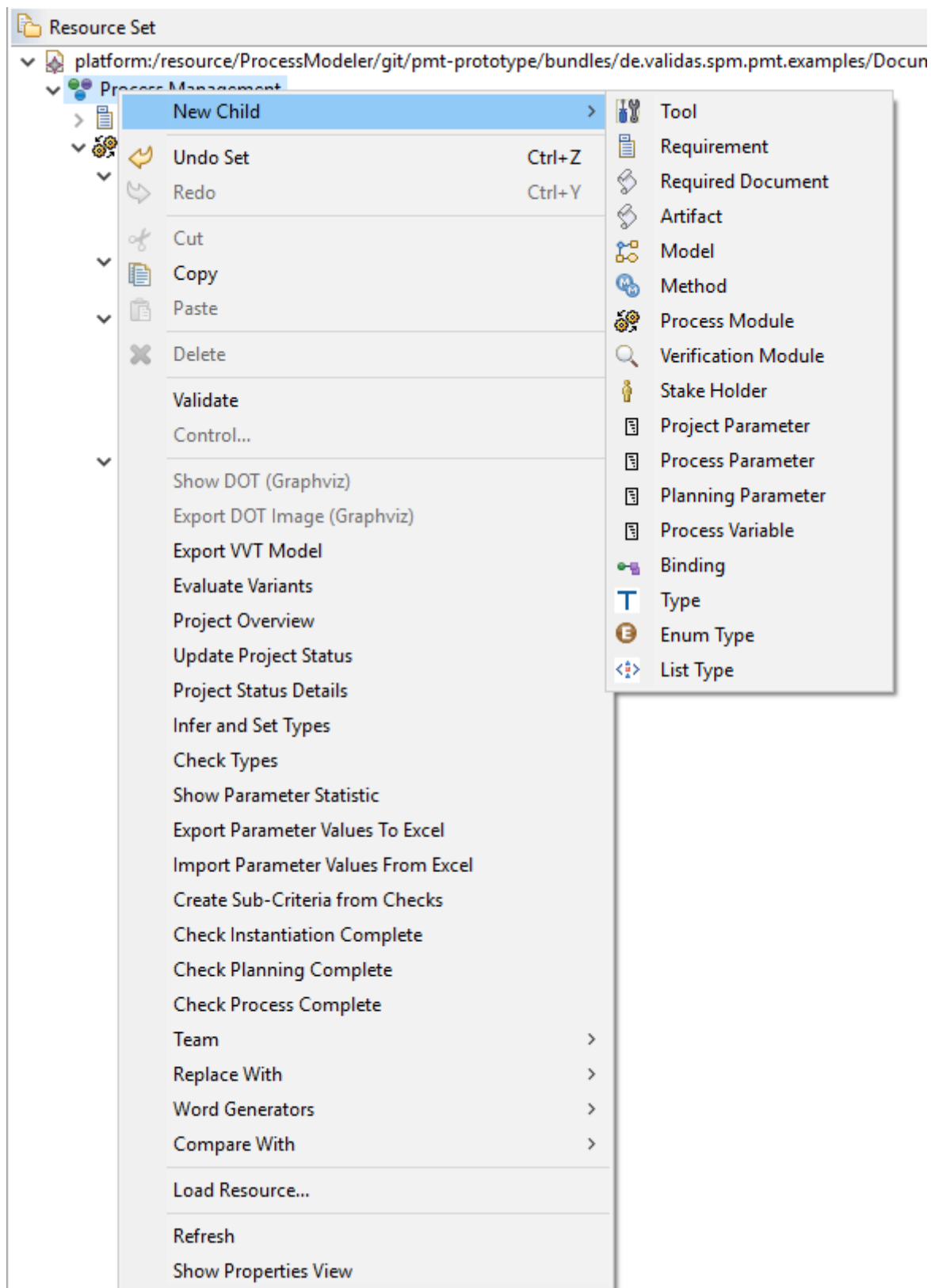


Figure 26: Tree Browser

Figure 26 shows the tree browser and the popup menu of the right mouse button click. It allows to add new elements to the model using the function "**New Child**". The children that can be added depend on the

selected model element and on the activated extensions. The available elements and their structure are explained in Section 9.

Furthermore, also other actions can be started using the popup menu, for example the report generation, export, import etc. Some actions support multiple selection.

The following popup actions can be activated by right-clicking on the elements in the browser. Note that some actions are gray. They can only be activated if the corresponding model extensions are activated.

7.2.1 On Various Elements

The following actions are applicable on several elements and have comparable behavior:

- **New Child:** Create a new child element. See Section 9 for the meta model that describes the possible elements.
- **New Sibling:** Create a new sibling element. See Section 9 for the meta model that describes the possible elements.
- **Cut:** Cut the selected element from the browser into the copy buffer.
- **Copy:** Copy the selected element into the copy buffer.
- **Paste:** Paste the element from the copy buffer into the selected element.
- **Delete:** Delete the selected element(s).
- **Validate:** The validate action is described in Section 8.5.1.

Furthermore there are some PMT specific actions available on several elements:

7.2.1.1 Show DOT (Graphviz)

This action is available on the following elements:

- ProcessModule
- Artifact
- Stakeholder
- Requirement
- Compliance

It shows the source code that is used to compute the Process and the compliance view, depending on the selected element as shown in Figure 27. It can be used for example in <http://www.webgraphviz.com/> to modify or debug the graphs.

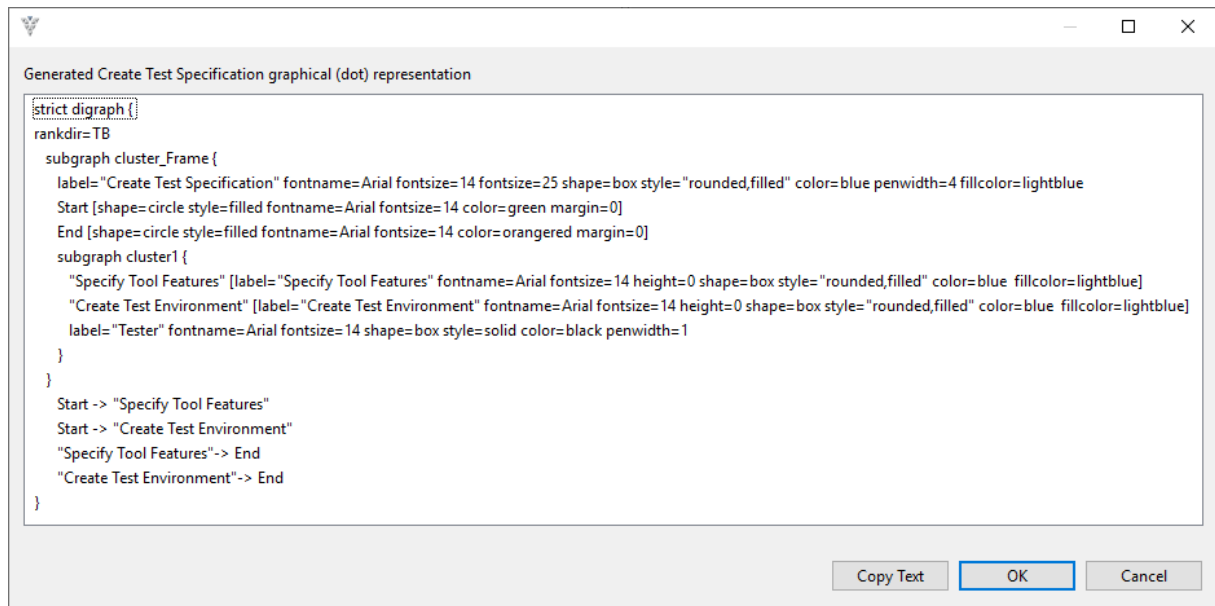


Figure 27: DOT Representation of the Graphical View

7.2.1.2 Export DOT (Graphviz)

Similar to the above “Show DOT (Graphviz)” actions in previous section, except that it saves the text into a file.

7.2.1.3 Infer and Set Types

Parameters, Variables and Terms have types that should be specified to ensure correct validation and evaluation of the terms. This can be done automatically using the PMT action “Infer and Set Types”, which is available on all “Named” elements and all “Terms” and “Bindings”. Once started (see Figure 28), it will create a textual report on which inferences it did as shown in Figure 29.

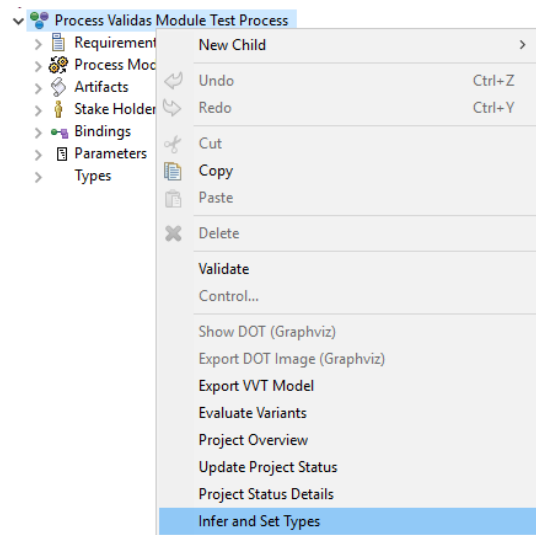


Figure 28: Start “Infer and Set Types”

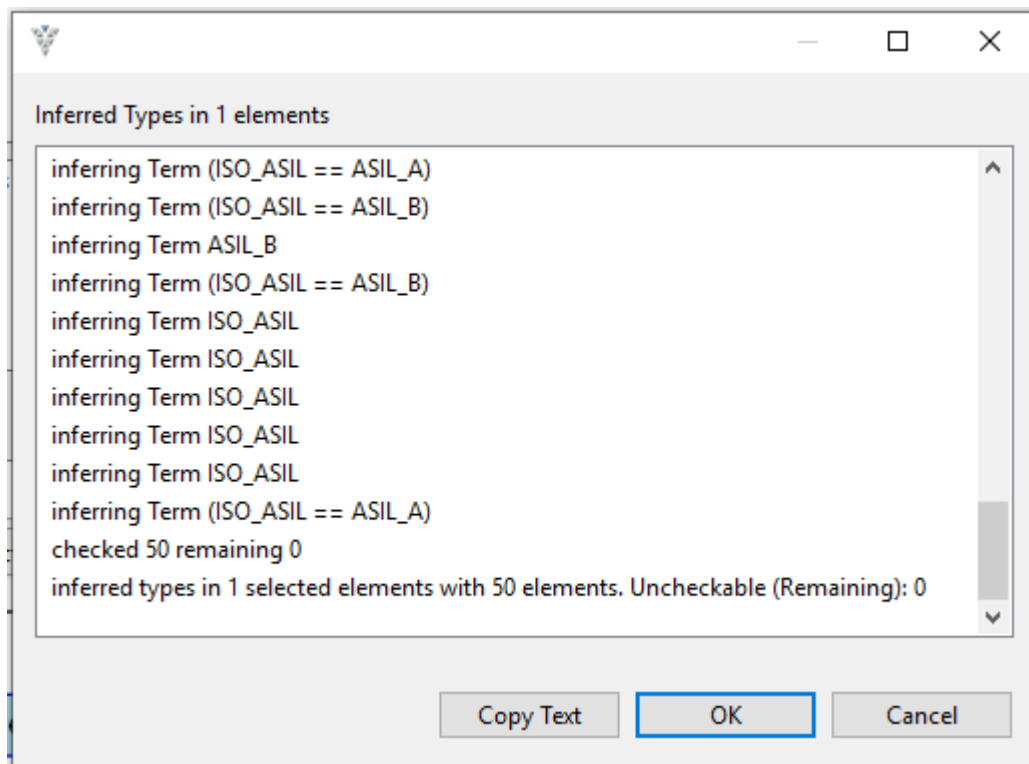


Figure 29: Result of “Infer and Set Types”

7.2.1.4 Check Types

The “Check Types” action can be started on all elements and checks the types in the selected element(s) and in all contained elements.

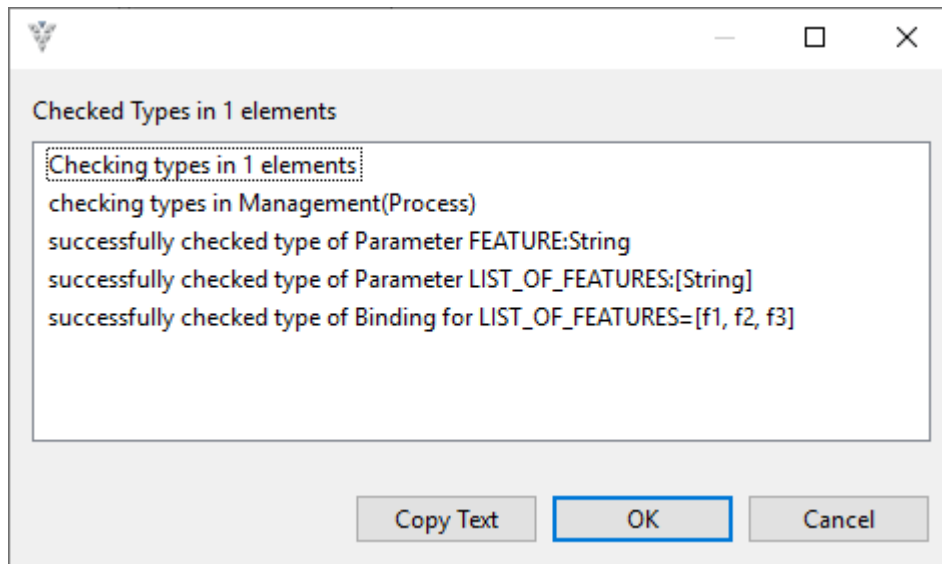


Figure 30: Result of type checking

7.2.2 On Process Element

On Process elements the following popup-actions are available (see Figure 31):

The following actions are Project specific:

- Check Project Status: computes the project status, see Section 8.2.11.
- Export Parameter Values To Excel: Exports the parameter values to excel, allowing them to be changed in Excel
- Import Parameter Values from Excel: imports parameter values from Excel and creates (or updates) binding for the parameters.

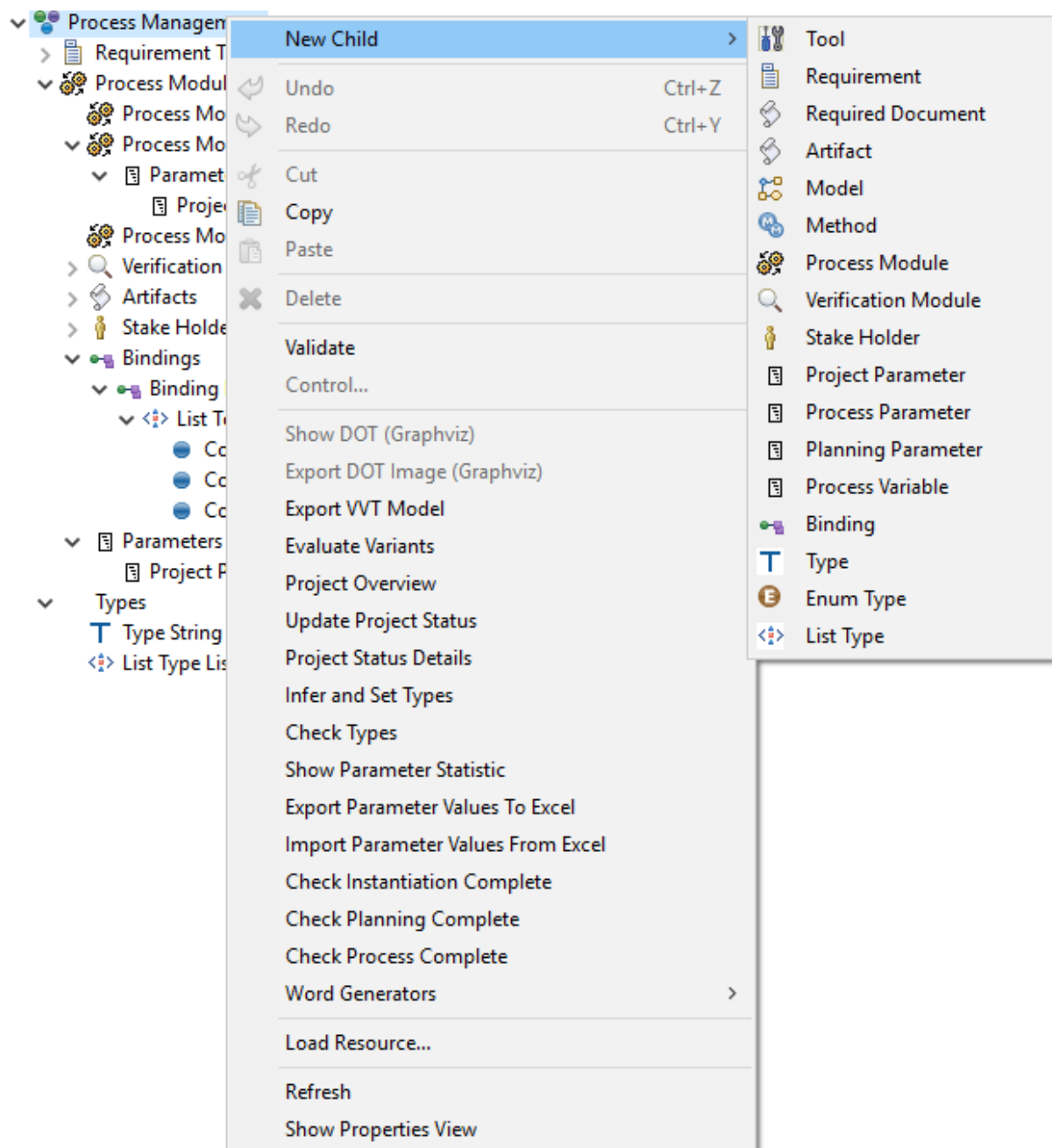


Figure 31: Popup Menu on Project

7.2.3 On ProcessModule Element

On ProcessModule elements the following popup-actions are available (see Figure 31):

The following actions are Project specific:

- Check Project Status: computes the project status, see Section 8.2.11.
- Reset Project Specific Data: Resets project specific data (ProjectRelevant, ProjectComment, Effort, NumberOfInstances, PlannedStartDate, PlannedEndDate, EndDate) to default values.

- Export Development Interface (Excel), see Section 8.6.3.4
- Export Offer (Excel + Word), see Section 8.6.3.5
- Export Parameter Values To Excel: Exports the parameter values to excel, allowing them to be changed in Excel
- Import Parameter Values from Excel: imports parameter values from Excel and creates (or updates) binding for the parameters.

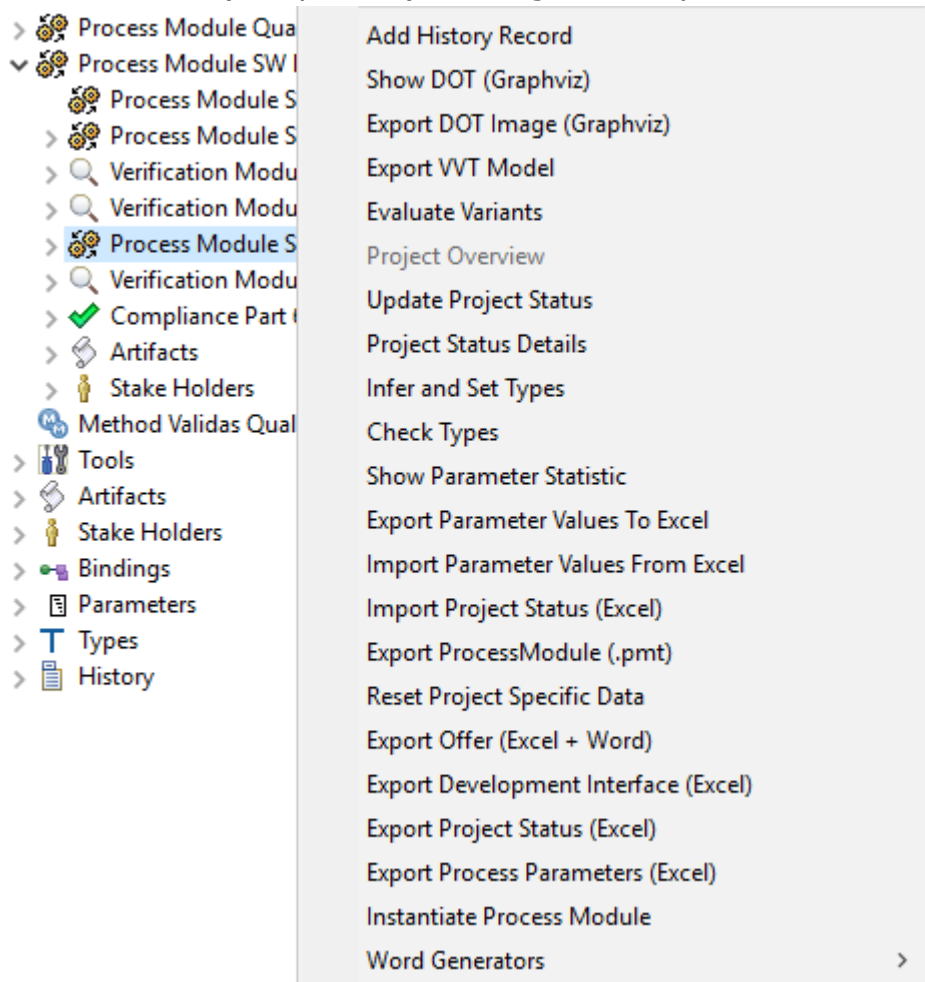


Figure 32: Popup Menu on ProcessModule

7.2.4 On Model Element

On Model elements the following popup-actions are available (see Figure 123):

The following action is Model specific:

- Import Ecore Model: This allows to import ecore models to support modeling of model-based processes, see Section 8.2.2.

Property View:

The property view shows the properties of the element that is selected in the tree browser. There are two forms of property views in PMT

- 1) The classical EMF view, which shows one line for each property (see Figure 34)
- 2) An EMF Forms based with groups of properties that can be collapsed (see Figure 35)

Using the small icon on the right top, the views can be switched (see Figure 33).

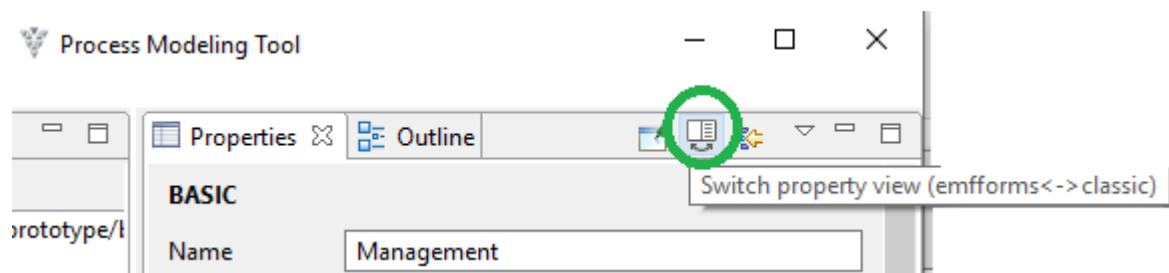


Figure 33: Switch Property Views

Property	Value
Name	Management
Description	Mini Example for QKit Planning
ID	
Comment	
Long Description	
Deactivated	false
Maximal Safety Level	ASIL_D
Preferences	
Filter Scope	

Figure 34: Classical EMF Property View (of Process)⁵

⁵ Note: Issue#39 states a problem with EMFForm of Process, that is not collapsing as expected

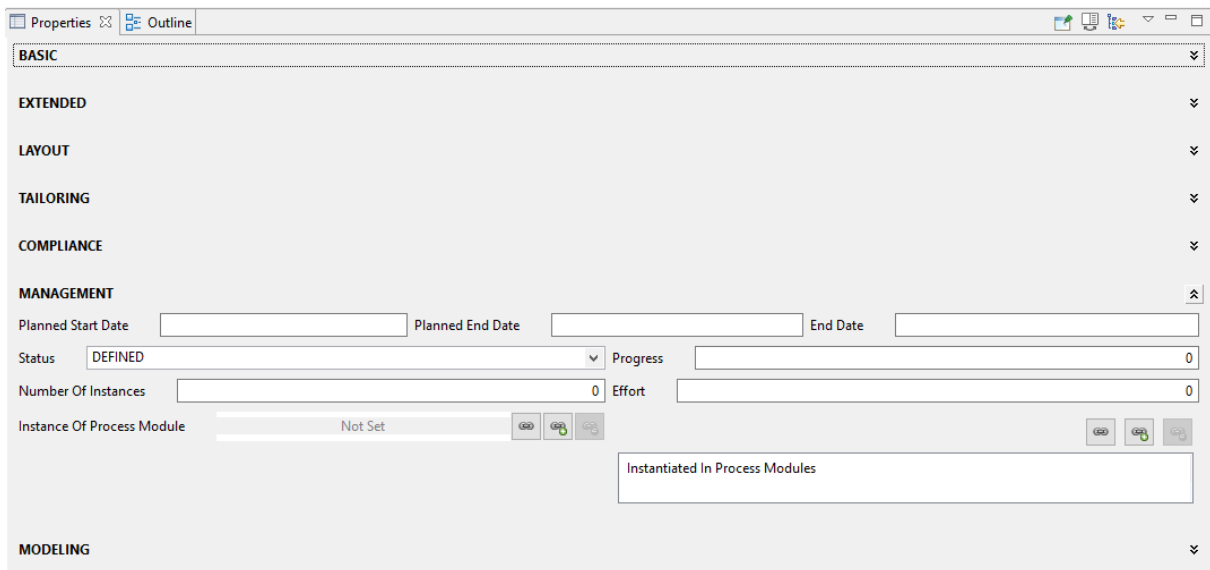


Figure 35: EMF Forms Property View of Process Module

The EMF Forms of all elements have the similar property categories that are represented by “collapsible groups”, see Figure 35.

- **BASIC:** this category contains the most frequently used basic properties like name description of the element.
- **EXTENDED:** less frequently used properties that do not fall under a specific other category
- **LAYOUT:** properties to impact and customize the generated layout of the process and compliance view
- **TAILORING:** All properties required for automated tailoring: Parameters, Bindings, but also types and their declarations
- **COMPLIANCE:** properties for the compliance with safety standards or requirements
- **MANAGEMENT:** properties to manage the process, e.g. efforts, dates, states,...
- **MODELING:** properties for model-based processes, e.g. meta-models and process conditions.

7.2.5 On Requirement Element

In order to change the “Recommended From” and “Recommended To” properties of many requirements at once, on the Requirement the following actions are available: “Set Recommended From” and “Set Recommended To”. Both can be started using the right mouse popup menu as shown in Figure 36. Note that they work recursively for all contained requirements and it is manually required to save the model afterwards.

The resulting changes are listed in the result dialog as shown in Figure 38

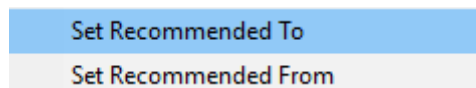


Figure 36: Starting Set Recommended Actions

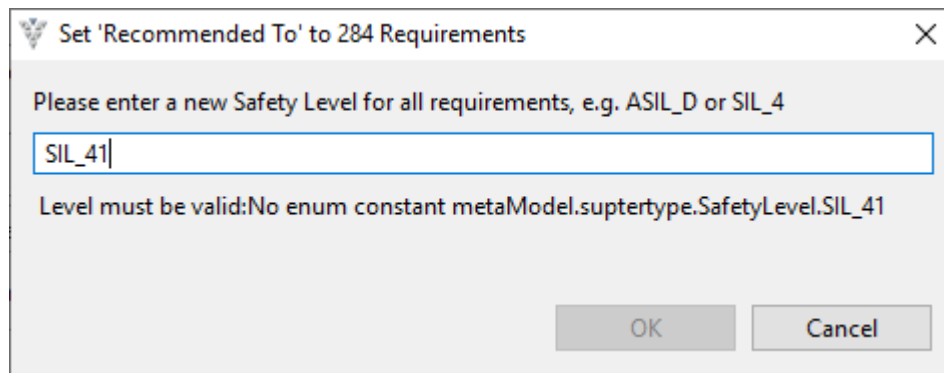


Figure 37: Enter New Safety Level Dialog

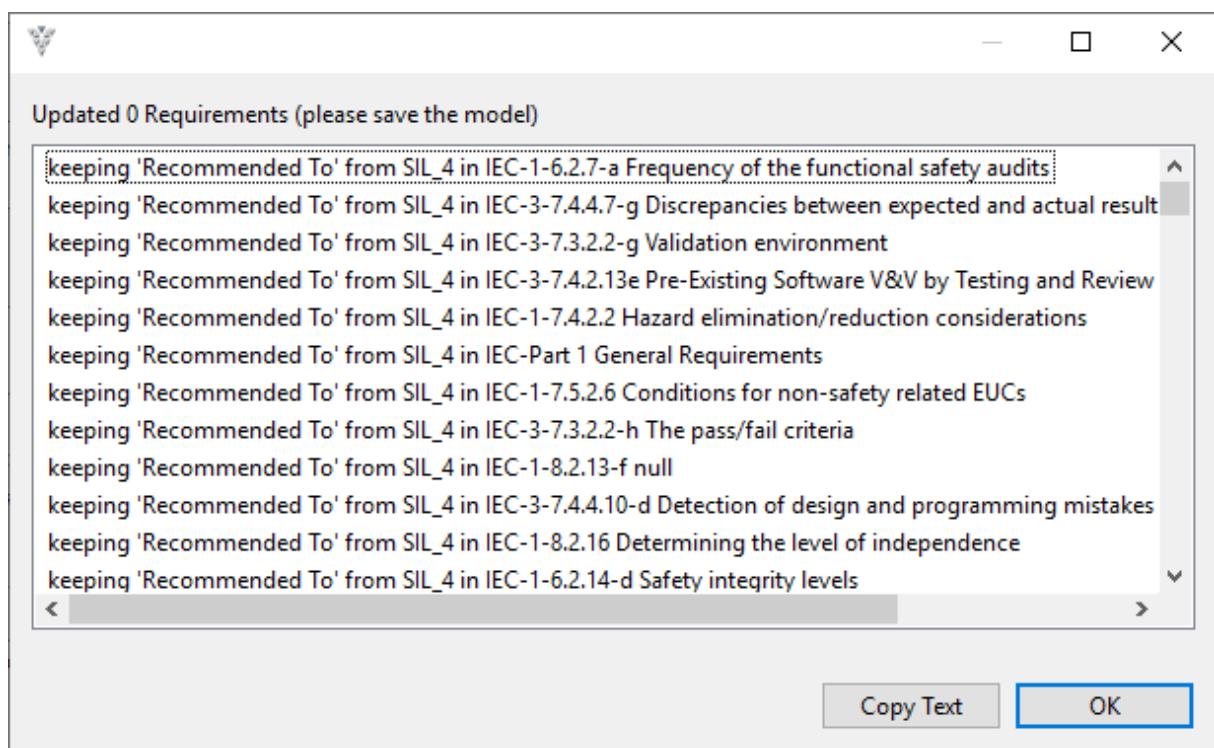


Figure 38: Result of Set Recommended Actions

Also available on Requirements: **Generate Compliance Structure** Action, see Figure 39 that generates a compliance template tree for the selected Requirement including the references to the requirement and containing an argumentation pattern that should be extended for the atomic requirements.

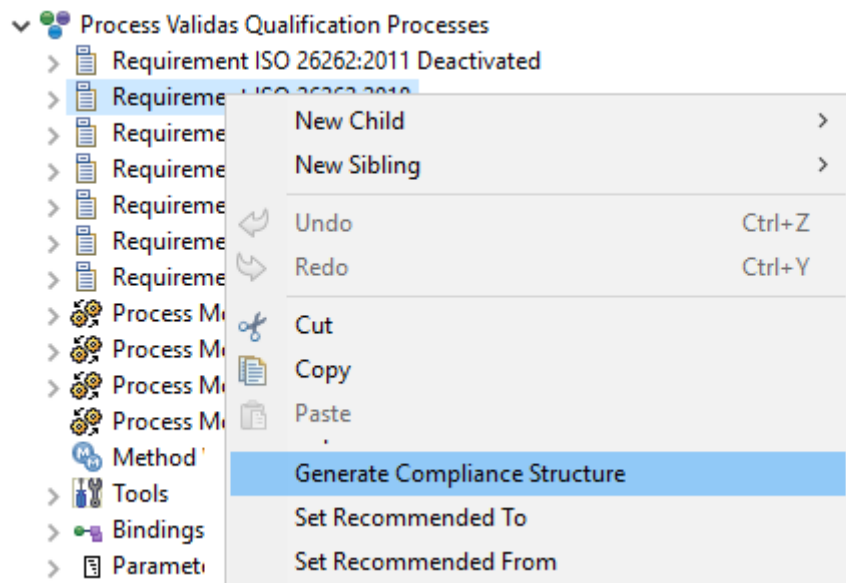


Figure 39: Generate Compliance Templates Action

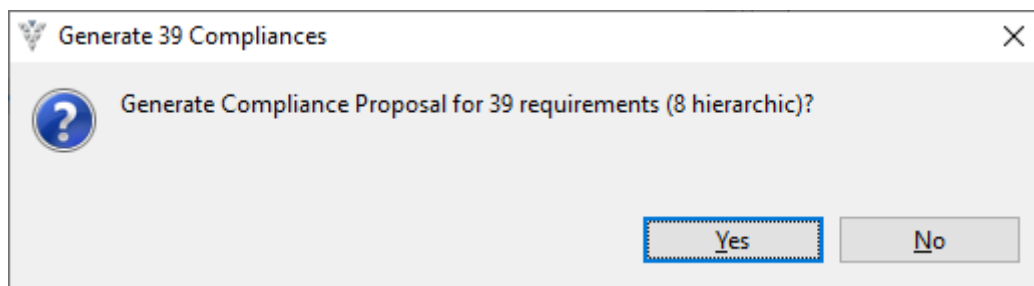


Figure 40: Confirmation of Compliance Structure Generation

After confirming the generation question (see Figure 40) the result (see Figure 41) is generated. This can be moved into the process modules for refinement.

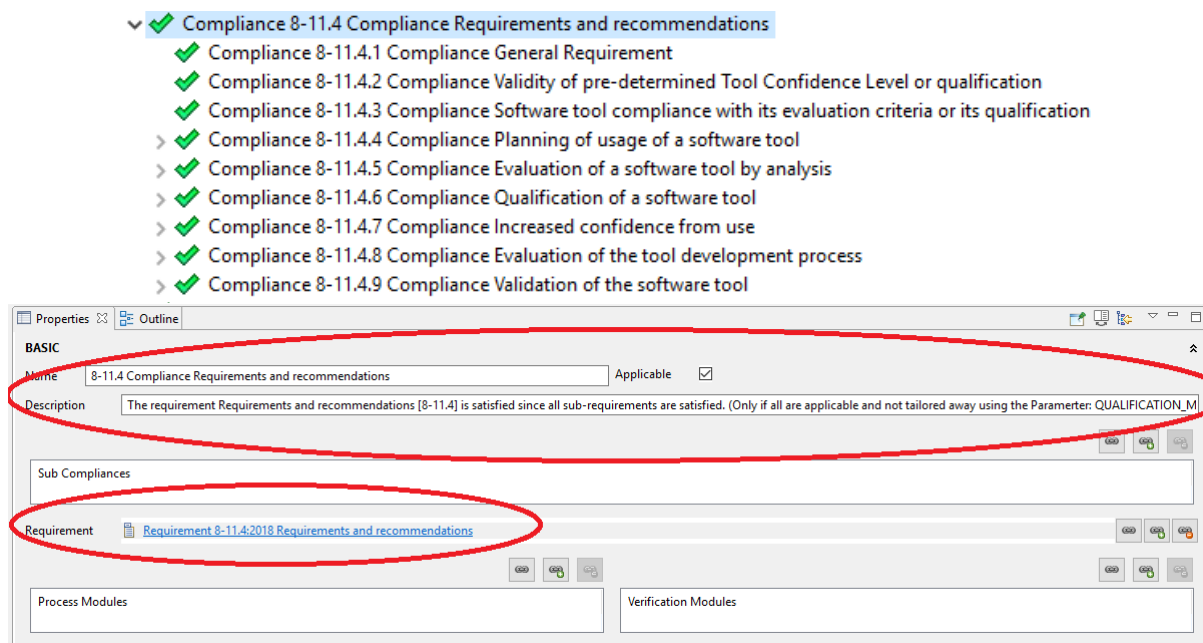


Figure 41: Generation Result Example

7.2.6 On Parameter, Binding, EnumValue and EnumType Elements

Since tailoring terms are not bi-directional linked, it is not clear, where they are used. For Example consider you have an EnumType TestResult = PASS / FAIL / ERROR and you want to determine in which elements (tailoring terms) it is used.

This finding of all references to a given Parameter, Binding, EnumValue or EnumType can be done with the so called "Show References" Action which is available on the references elements only.

The result of the action lists all occurrences of this element in the complete model (independent from tailoring). The action can be started using the right mouse popup menu as shown in Figure 42.

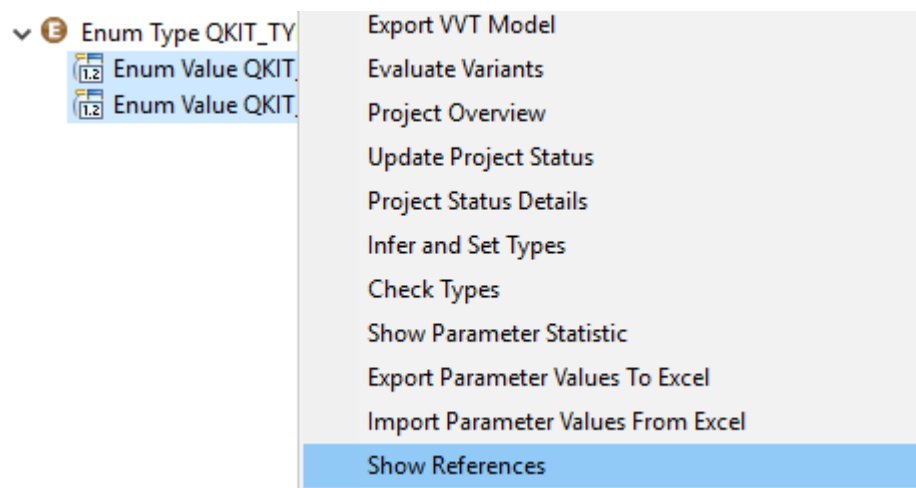


Figure 42: Starting the Show References Action

The result is displayed in a list text dialog as shown in Figure 43

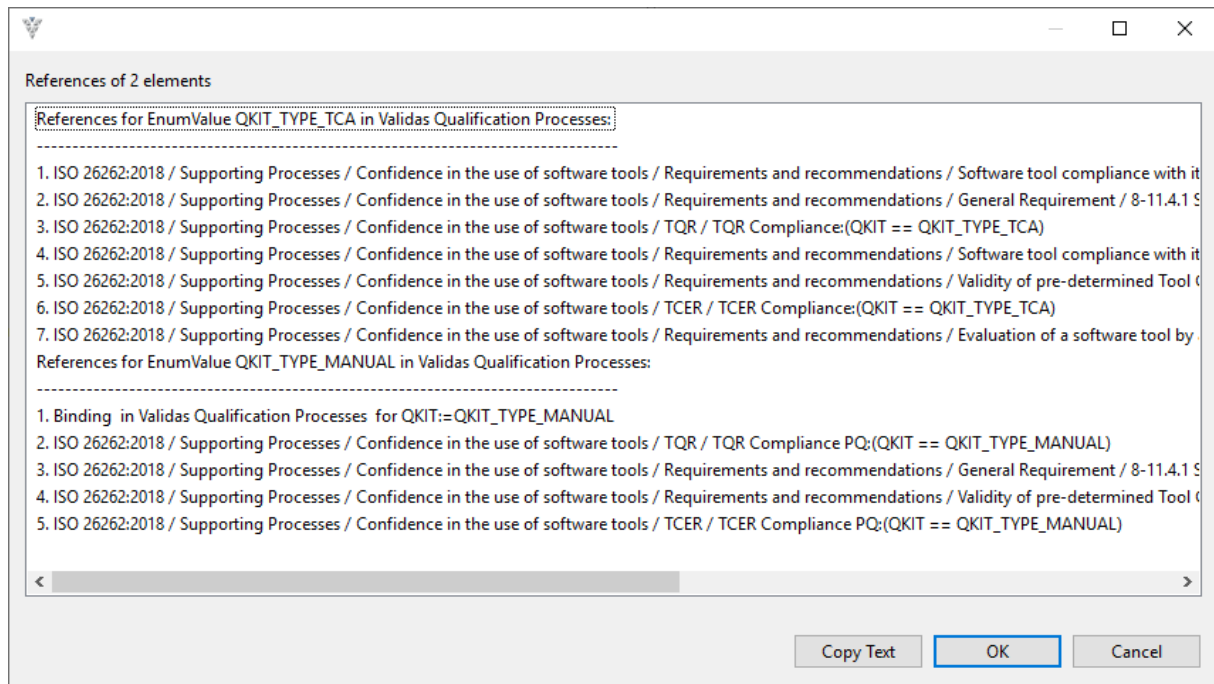


Figure 43: Result of Show References Action

7.3 Process Module View

The process module view shows the graphical representation of the process using the graphical notation defined in Section 5, depending on the current selection. It is working for "ProcessModules", "Artifacts" and "StakeHolders" and shows their process view.

Important to note is that it requires graphviz (dot.exe) to be in the path otherwise it will not work (see Section 6 for installation). Furthermore process modules require to have a **stakeholder assigned** in order that they can be graphically visualized, otherwise the graphic will just be ignored.

PMT generates the images using graphviz and stores them temporarily in files (.gv and .png) into a temporary file directory pointed by the variable "java.io.tmpdir"/PMT, so for example: C:\Users\oscar\AppData\Local\Temp\PMT. Here you can find also the images for further usage. However the preferred way to access the sources for graphviz is definitely the "Show DOT (Graphviz)" action that is available on the elements having a graphical view, see Sections 7.2.1.1 and 7.2.1.2.

7.4 Compliance View

The compliance view shows the compliance argumentation using the Goal Structured Notation (GSN), see <https://www.goalstructuringnotation.info/> for more information.

It does not show the complete tree of arguments, since most requirements are hierarchically and this would lead to too complex images. It just shows one hierarchy level. The generated compliance report contains all compliance views and is therefore complete.

Like in the process module view the graphs are generated in the temporary directory and can be exported using SHOW and export actions, see Sections 7.2.1.1 and 7.2.1.2.

7.5 Projection View

The projection view shows the result of model projection. Model projection is an information on the model or on selected parts of it. Model projection shows all projected values grouped by attributes and classes, such that it can be used as statistical information (e.g. the number of modeled processes) or to find a specific element by name.

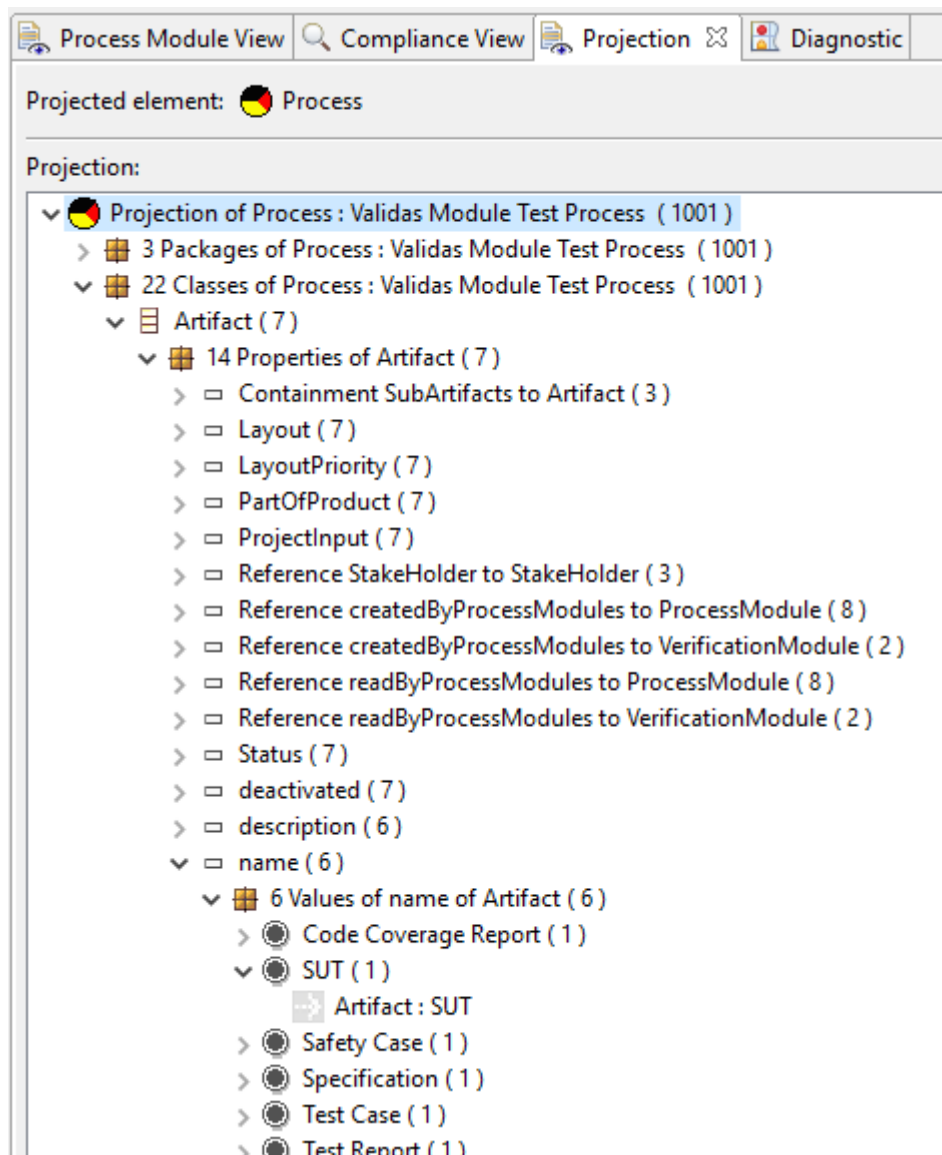





Figure 44: Model Projection View

The Model Projection, see Figure 44 view has a root (🇩🇪) for the selected element(s) that is/are projected. The root contains the type and the name of the projected element and the number of contained elements in brackets. Below there are the different projection groups (📦) listed:

- The package group: shows the projection split by packages
- The classes group: shows the projection split by the classes. In the classes group, every class is listed with the number of properties
- The properties group (if configured in the PMT model projection preferences). It contains all properties, independently from the classes, e.g. if an Element with a given property Name X is searched, it will be found there, independently if it is a Process, Artifact, Tool or any other element with this property.

Under every class (), the properties () are listed. There are two kinds of properties: attribute and relation properties. Attributes are listed by their names, e.g. "Comment", "Deactivated", Relations are listed by their type, e.g. "Containment Process to Tool" or "Reference to Stakeholder". Both have their quantities in brackets. Empty or "null" values/references are not counted. Unlike Attribute properties, relation Properties do not have values listed, but model projections () of the related elements, see Figure 45

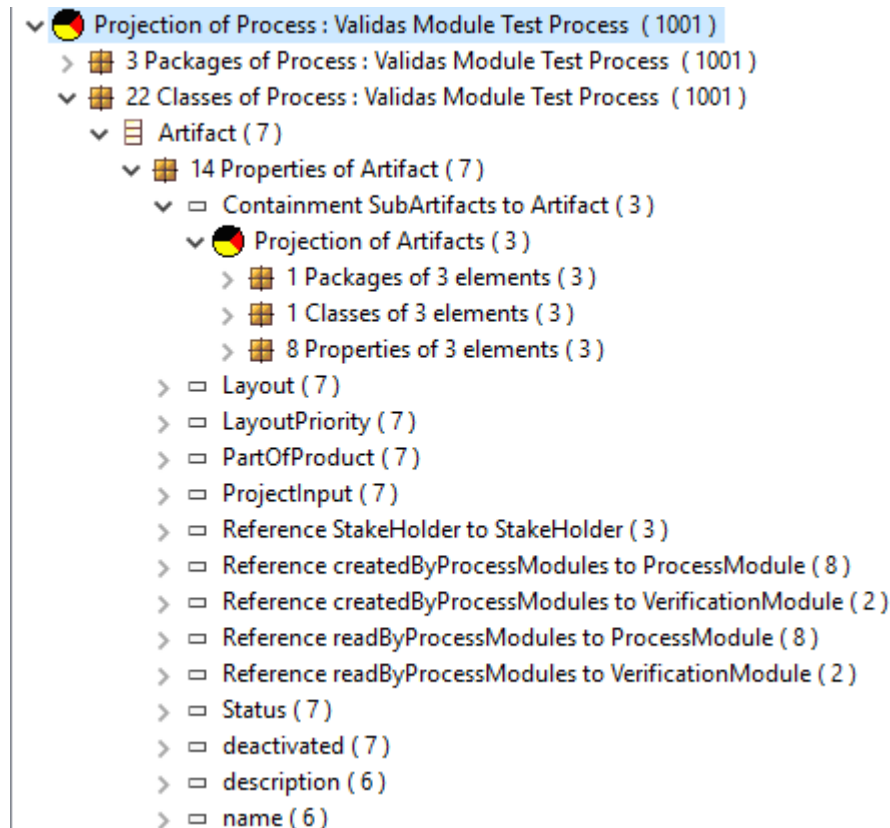




Figure 45: Model Projection of Relations

For every property, the values () are listed that have been found during projection, e.g. "true (13)" in Figure 47. Below that, the values pointers () to the occurrences are listed with their types and names, e.g. "MetaModelElement: tcm.Attribute". The pointers can be used with a double click to navigate in the browser to the elements with that value. On the right upper corner there is an icon that can be used to freeze the current model projection. Otherwise model projection will always be recomputed based on the current selection in the browser.

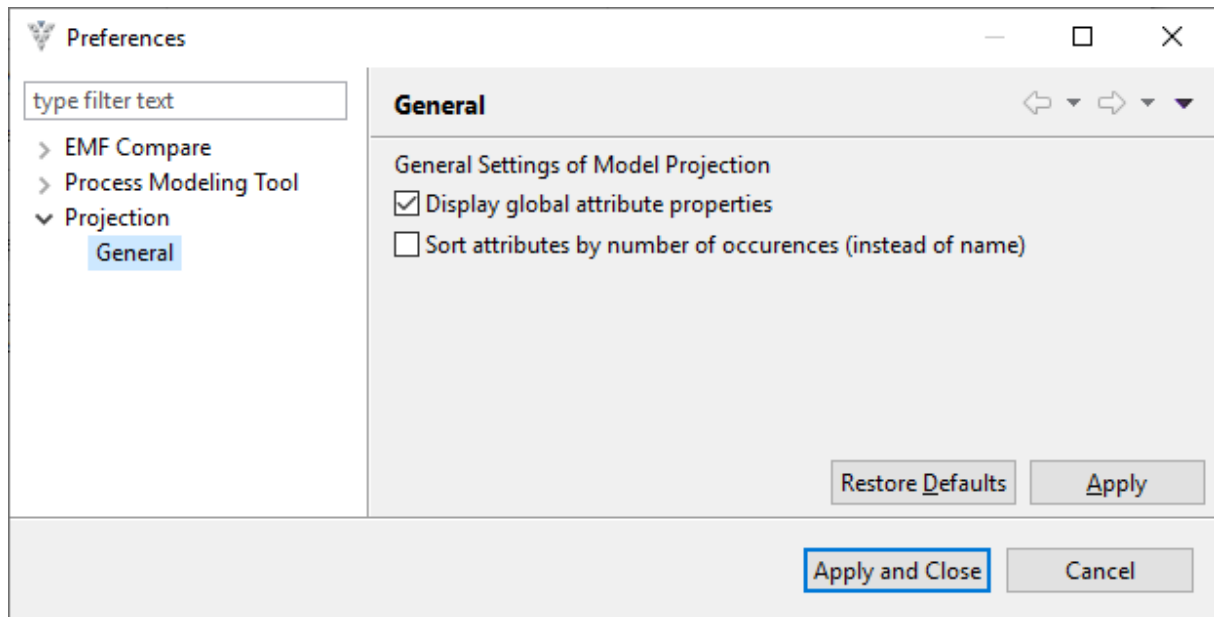


Figure 46: Model Projection General Preferences

Model projection can be configured in the preference pages, see Figure 46.

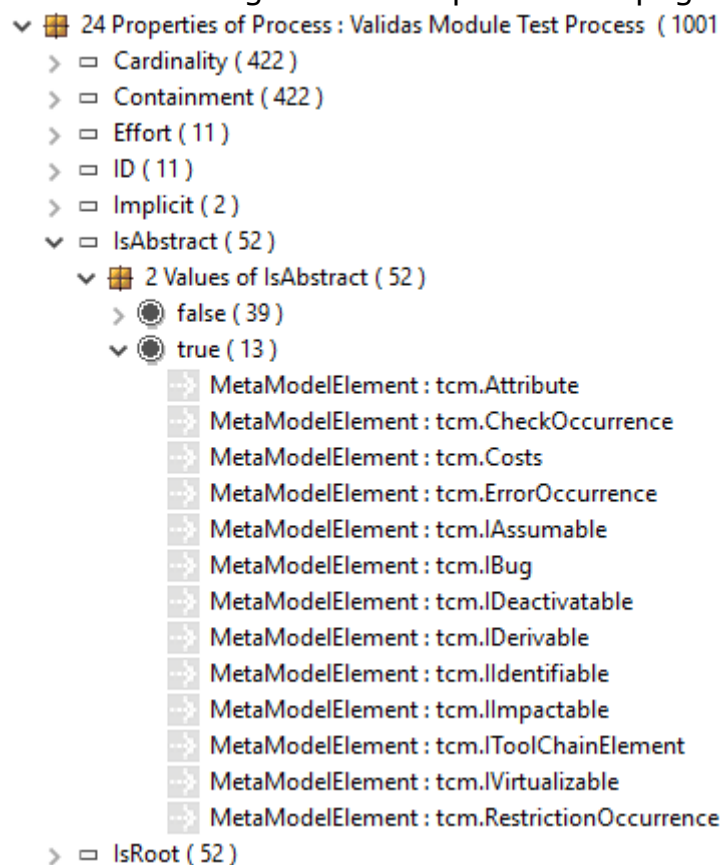


Figure 47: Global Attribute Property Projection Group

It supports the following two settings:

- Display global attribute properties: If this is selected, there will be a global attribute properties projection group in the root of the projection, see Figure 47. It contains all attribute properties of all classes.

- Sort attributes by number of occurrences: If this is selected, the sorting in the model projection view is changed accordingly. Default sorting is alphabetically, but it can be changed to list the most frequently found elements first (in a descending order).

7.6 Diagnostic View

The diagnostic view (see Figure 48) shows the results of the last performed model validation. It can be used to navigate to the elements that violates a rule.

Note it is not automatically updated, except when running validation.

Compliance View Projection Diagnostic Process Module View				
48 errors, 0 warnings, 0 others				
Model Element	Rule	Description	Qualified Name	Resource
▼		Other (48 items)		
Artifact	Artifacts are used	Artifact Specification: No creating/reading process modules found (an...	Artifact Specification	Management.pmt
Artifact	Artifacts are used	Artifact Test Environment: No creating/reading process modules foun...	Artifact Test Environment	Management.pmt
Artifact	Artifacts are used	Artifact Test Implementation: No creating/reading process modules fo...	Artifact Test Implementation	Management.pmt
Artifact	Artifacts are used	Artifact Test Specification: No creating/reading process modules foun...	Artifact Test Specification	Management.pmt
Artifact	Artifacts have Creator defined	Artifact Specification: no active Creator and not project input.	Artifact Specification	Management.pmt
Artifact	Artifacts have Creator defined	Artifact Test Environment: no active Creator and not project input.	Artifact Test Environment	Management.pmt
Artifact	Artifacts have Creator defined	Artifact Test Implementation: no active Creator and not project input.	Artifact Test Implementation	Management.pmt
Artifact	Artifacts have Creator defined	Artifact Test Specification: no active Creator and not project input.	Artifact Test Specification	Management.pmt
Artifact	Artifacts have Readers/Verifiers defined	Artifact Intro: no active Readers/Verifiers and not part of product	Artifact Intro	Hierarchy.pmt
Artifact	Artifacts have Readers/Verifiers defined	Artifact Specification: no active Readers/Verifiers and not part of prod...	Artifact Specification	Management.pmt
Artifact	Artifacts have Readers/Verifiers defined	Artifact Test Environment: no active Readers/Verifiers and not part of p...	Artifact Test Environment	Management.pmt
Artifact	Artifacts have Readers/Verifiers defined	Artifact Test Implementation: no active Readers/Verifiers and not part ...	Artifact Test Implementation	Management.pmt
Artifact	Artifacts have Readers/Verifiers defined	Artifact Test Specification: no active Readers/Verifiers and not part of p...	Artifact Test Specification	Management.pmt

Figure 48: Diagnostic View of Last Validation Results

8 Features of PMT

PMT has the following features that are described in this section:

- Terms and Types
- Process Modeling
- Validation
- Interfaces
- Report Generators
- Preferences
- Scoping

8.1 Terms and Types

PMT is based on a mathematical model of **terms** allowing to automatically check consistency and to automatically evaluate terms, for example to tailor variant terms and conditions in the process model. The model is inspired by the second order lambda-calculus consisting of terms and types, being constructed by constant, functions and variables.

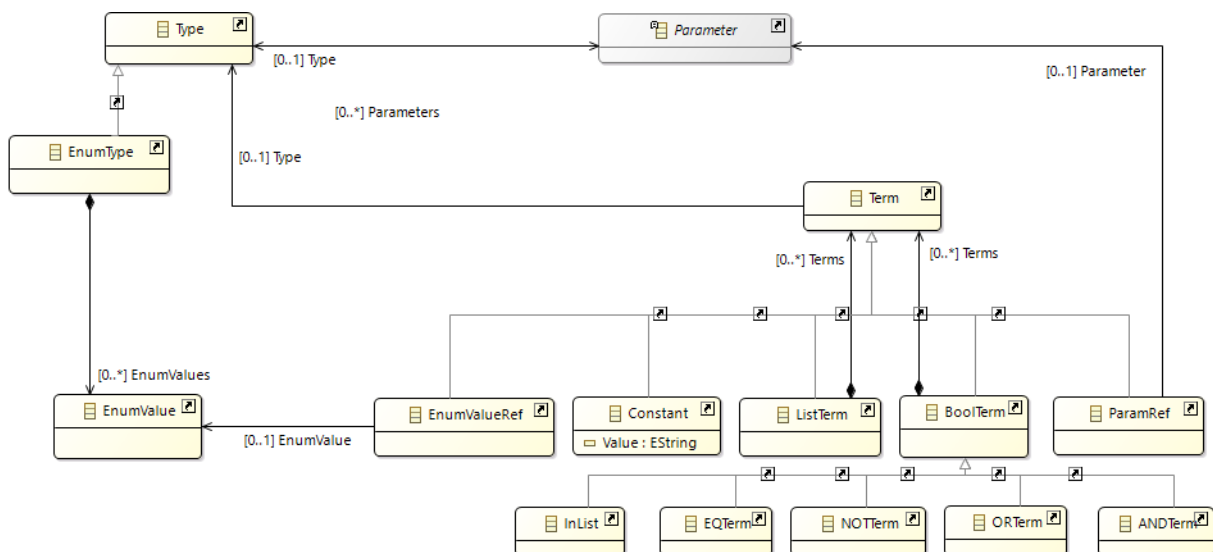


Figure 49: Terms in PMT

While Figure 49 looks quite complex, terms are very simple. They can consist only of the following elements

- Constants
- Boolean and list terms
- References to modeled elements

Boolean terms can be created in the allowed places (Variantable elements) in the tree browser by inserting new elements using the "New Child" action (or by copy & pasting existing terms). Note that the terms

have to be created according to their logical structure, starting with the root node. For example if a disjunction term shall be created it has to be done as shown in Figure 50. The arguments for all terms are specified in the meta-model section 9.5.

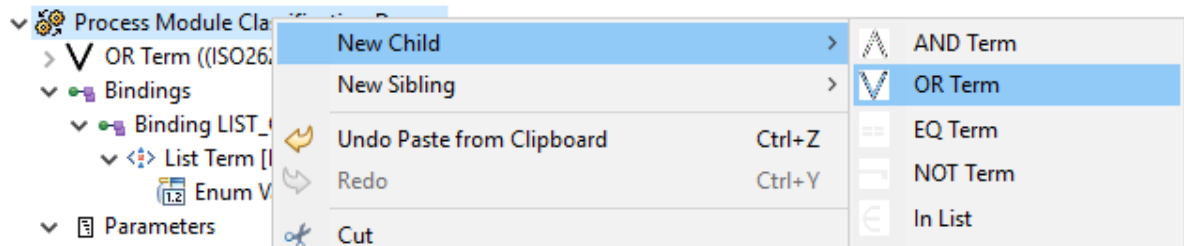


Figure 50: Creating new Terms

We demonstrate terms using an example term / condition:

ISO26262 ∈ LIST_OF_STDS || IEC61508 ∈ LIST_OF_STDS

This term is modeled as depicted in Figure 51: The tree-browser shows the tree structure of the term: It consists of an OR-Term with two similar sub-terms, constructed by an "In List" (∈) Boolean operator and two arguments: first a reference to a value of an enumeration and a reference to a process parameter "LIST_OF_STDS".

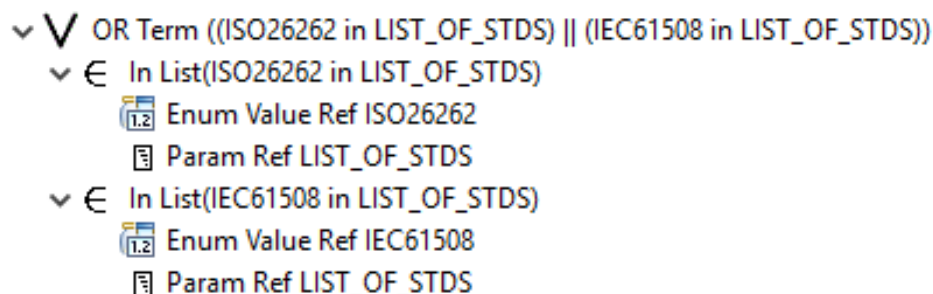


Figure 51: Example Term

This example shows that the term

ISO26262 ∈ LIST_OF_STDS || IEC61508 ∈ LIST_OF_STDS

is a boolean term, consisting of two sub-terms and that ISO26262 and IEC61508 are enumerated values that are referred within this term as well as the list of standards, which is a reference to process variable.

In order to evaluate the term the value of the process variable LIST_OF_STDS has to be defined. This can be done by a so called "Binding", that assigns ("binds") a concrete value to the variable.

Bindings can be contained in Processes (global) and ProcessModules (local). They have a reference to a parameter (to which the values are assigned) and they contain the value, see Figure 52.

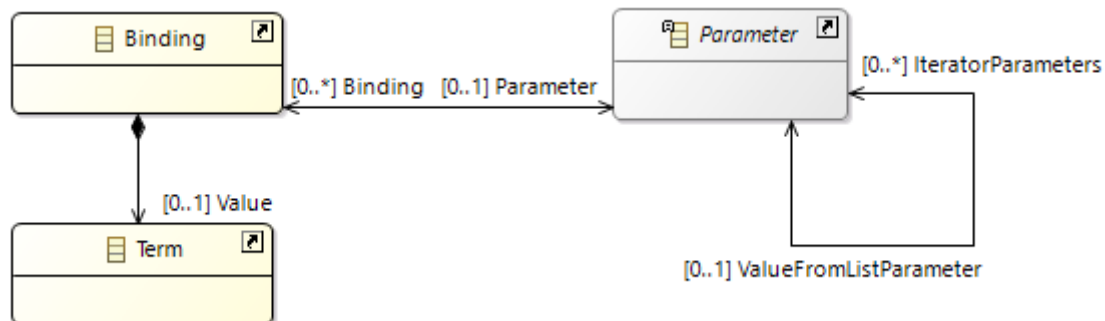


Figure 52: Binding Model

The bindings are displayed in PMT tree browser, especially the contained values, see Figure 53. Note that in the above example the value for the parameter "LIST_OF_STDS" is a list value (constructed by the "ListTerm" Term) with one value (reference to an EnumValue, as explained above).

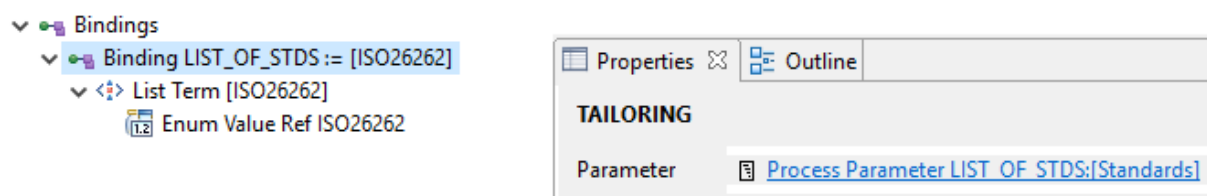


Figure 53: Binding Representation in Tree Browser and Property View

The overall example of terms (including definition and binding) looks as shown in Figure 55 in the tree browser (It is also contained in the Example/Documentation/Terms.pmt file).

In addition to the Variant-Term and the Binding it also contains the definition of the used types:

- 1) The EnumType Standards (with all EnumValues as children)
- 2) The ListType "ListOfStandardTypes", which contains a reference to the "BaseType" of the list ("Standards"), see Figure 54.

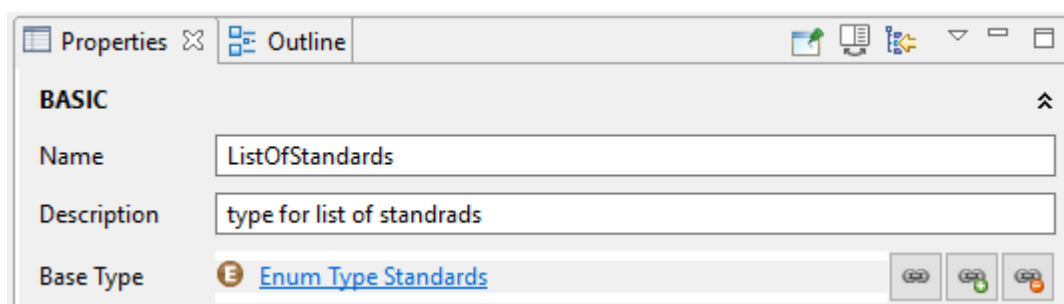


Figure 54: Property View of ListType

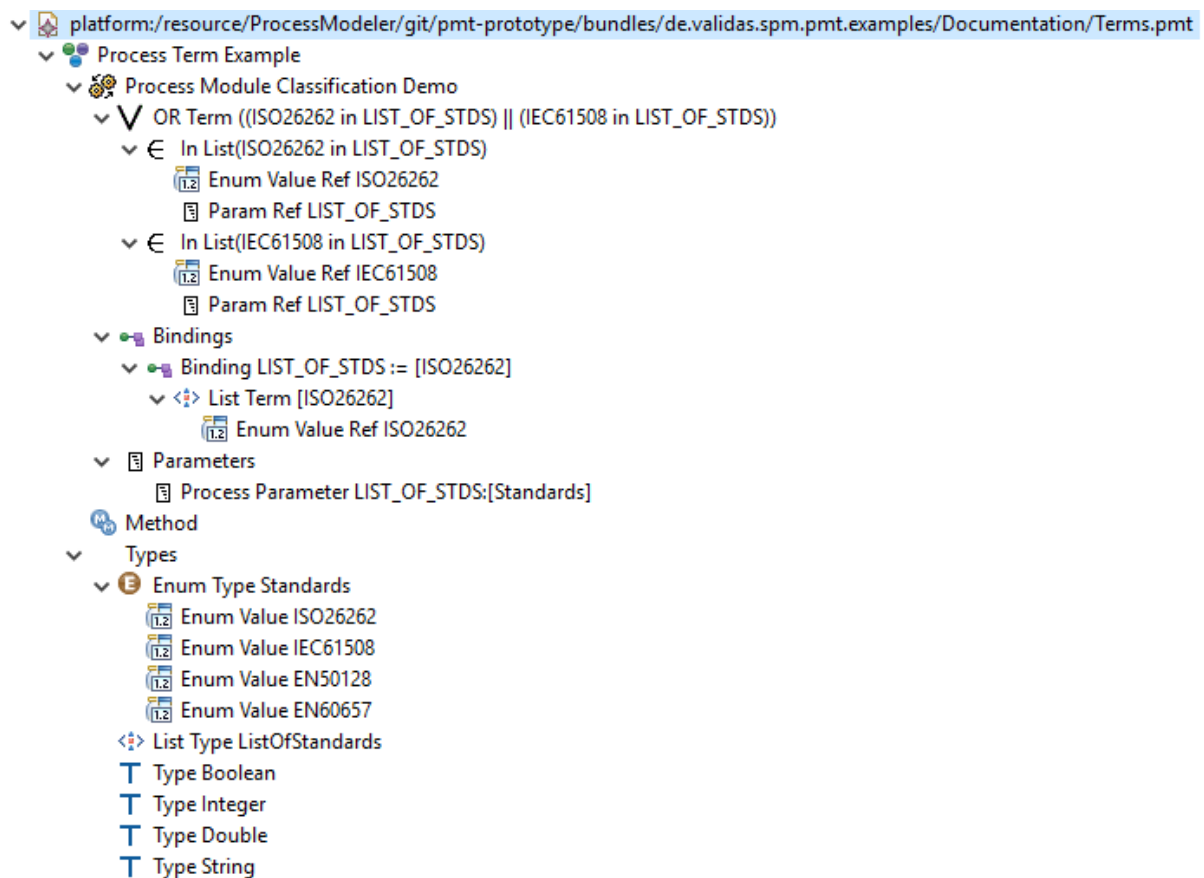


Figure 55: Complete Term Example

The **types** in PMT consist of three different types (see Figure 56):

- Base Types (like Boolean, Integer, Double)
- Enumeration Types: with defined constant values (see “Standards” in above example)
- List Types: describe the type of lists over base types (see “ListOfStandards” in above example).

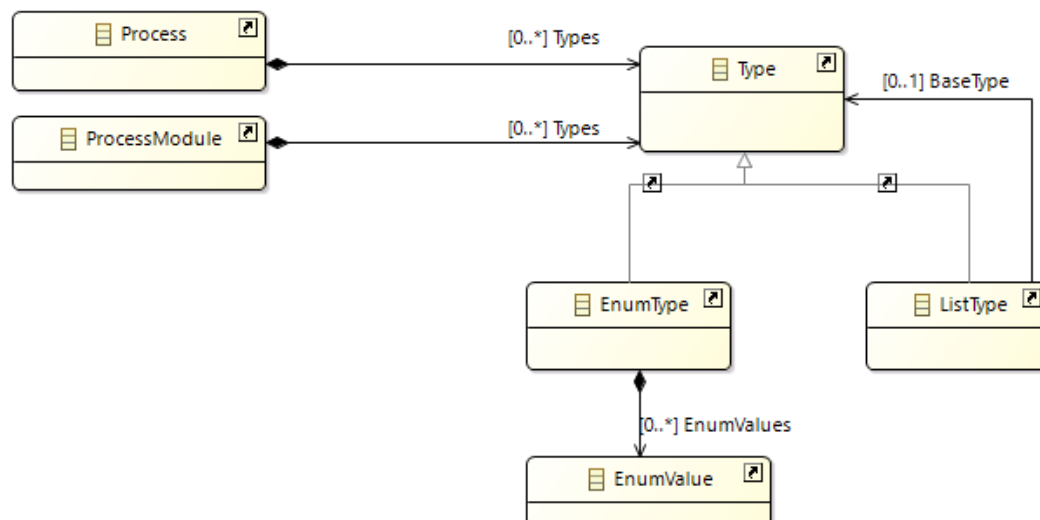


Figure 56: Model of Types

Figure 56 also shows the place in which types can be defined:

- Process: contains global types
- ProcessModule: contains local types

Note: if PMT infers types it also creates the default global types: "String", "Boolean", "Integer" and "Double" (see Figure 56). The corresponding List-types are not automatically generated and have to be created manually from the user in case they are needed.

Evaluating terms means computing their values. This is only possible for terms that have no "unbound" variables, e.g. parameters without bindings. PMT evaluates terms according to the following rules:

- Constants are evaluated to themselves, i.e. constants are already values.
- Parameters are evaluated to the evaluation of the bound terms. PMT searches for bindings local first and then going upwards until the global Bindings are considered.
- If parameters have no bound values they cannot be evaluated, this might occur for example in a parameterized process that has not been tailored by binding the variables.
- List-Terms are evaluated by evaluating all elements in the list and creating a new ListTerm with the evaluated arguments.
- InList terms are evaluated by evaluation of both arguments and then creating a result Constant with value True if the term is in the list, otherwise False.
- Boolean Terms are evaluated by lazy evaluation (but do not catch exceptions due to un-evaluatable arguments)
 - EQTerm(x,y) evaluates to TRUE, if x and y evaluate to identical terms.
 - NOTTerm(x) evaluates to TRUE, if x evaluates to FALSE and vice versa.
 - ORTerm(x,y) evaluates to TRUE, if one argument evaluates to TRUE (Evaluation order is left to right), otherwise to FALSE.
 - ANDTerm(x,y) evaluates to FALSE, if one argument does not evaluate to TRUE (Evaluation order is left to right), otherwise to TRUE.

Evaluation of terms can be triggered using the Evaluate Variants Action on Variantable elements, resulting into a dialog showing the evaluation results as shown in Figure 57.

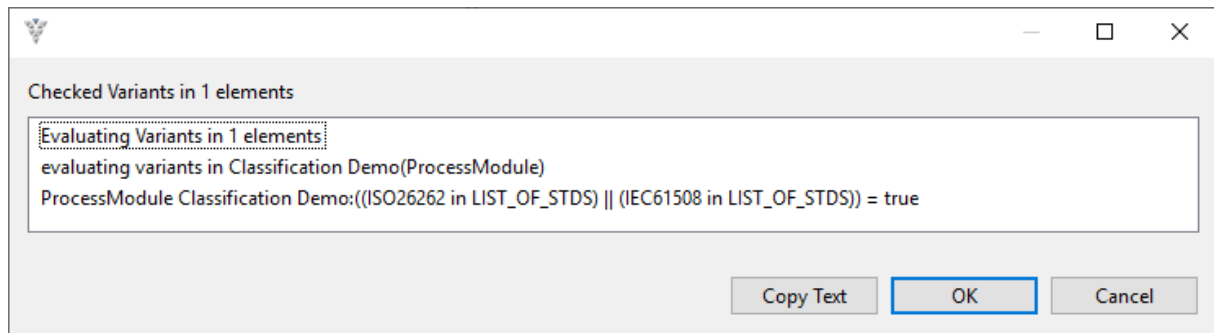


Figure 57: Variant Term Evaluation Result (Evaluate Variants Action)

8.2 Process Modeling

The process model consists of several aspects:

- Processes
- Requirements
- Model-Based Processes
- Compliance
- Reuse & Linking
- Tailoring
- Tools
- Project Management

8.2.1 Processes

Processes are modeled using the following elements

- ProcessModules: Describe activities / tasks, typically producing outputs from inputs. ProcessModules should be named starting with verbs, e.g. "Run Tests". Verification Modules are a special form of ProcessModules that perform verification activities to verify artifacts.
- Artifacts: Represent the data in the processes that are used as input / outputs from the ProcessModules
- StakeHolder: Are responsible for artifacts and process modules

There are two modes to describe a process

- Artifact-based
- Artifact-free

In the artifact-based modeling style every process module has to specify the used artifacts (input & output). The artifacts then define a sequence of the processes, see Figure 58.

In the artifact-free modeling techniques the user has to specify the sequence of processes, see Figure 59. Artifact-free modeling can only be done if the artifacts are implicitly known. For Safety Related Processes we recommend and use Artifact-based modeling, also because those

processes can be managed better, see the status of artifacts in Section 8.2.11.

Note that Stakeholders are always required in the model and without responsible stakeholder no process view can be generated.

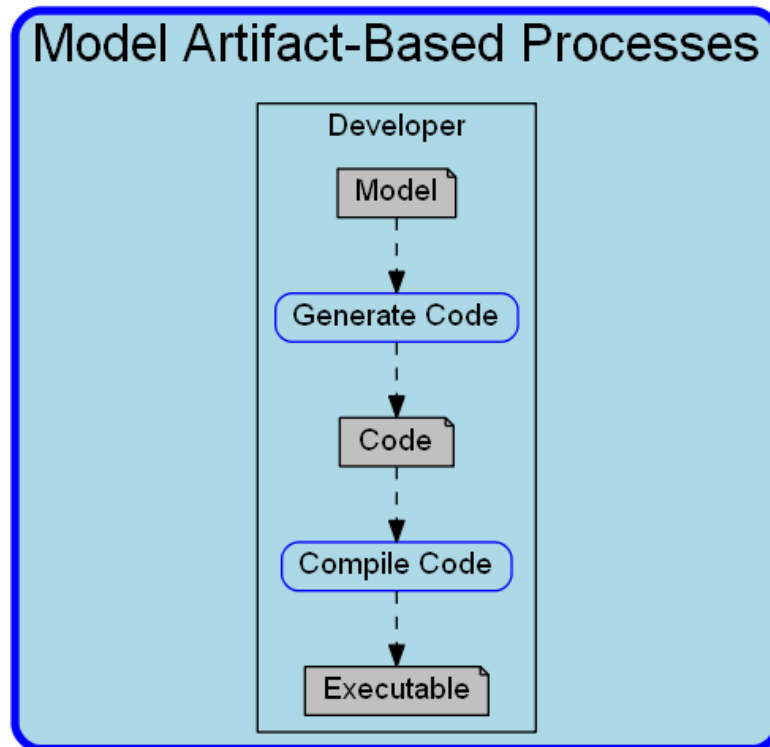


Figure 58: Artifact-Based Modeling

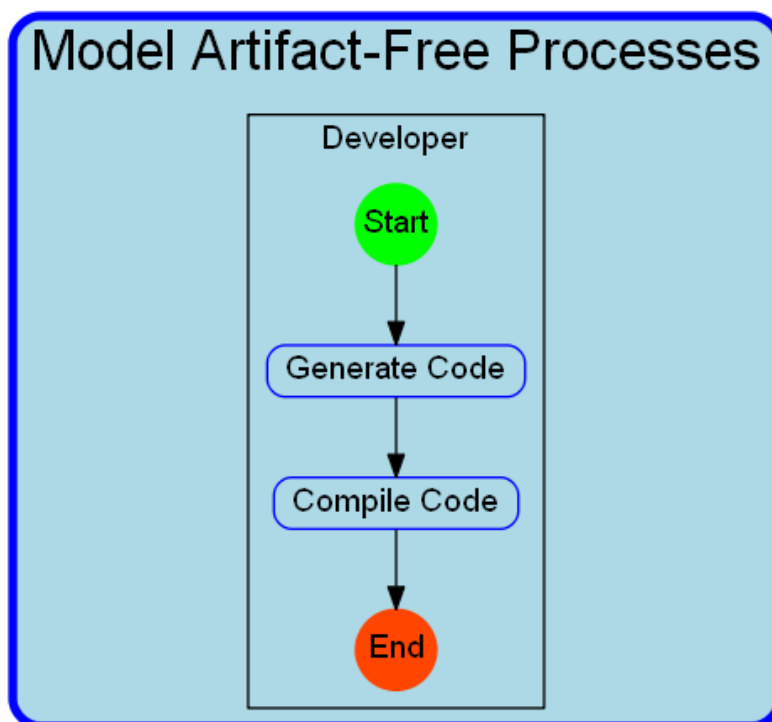


Figure 59: Artifact-Free Modeling

Note that ProcessModules can be nested hierarchically to keep processes manageable and viewable.

An important request, required by most safety standards is the consistency of the processes. To achieve this consistency we mainly require that the inputs and outputs of a hierarchic process modules are used and created by corresponding sub-processes. This means that all artifacts that are used/produced somewhere within the process have to be input/output of the process (except temporary artifacts). In order to avoid numerous inputs/outputs of processes, artifacts can be grouped hierarchically as well.

Consider the example in Figure 60: The left part shows the inner view of the process “Main” that creates two outputs from two inputs by two sub-processes, the right part shows the outer view of the process “Main” with all inputs/outputs of the inner view.

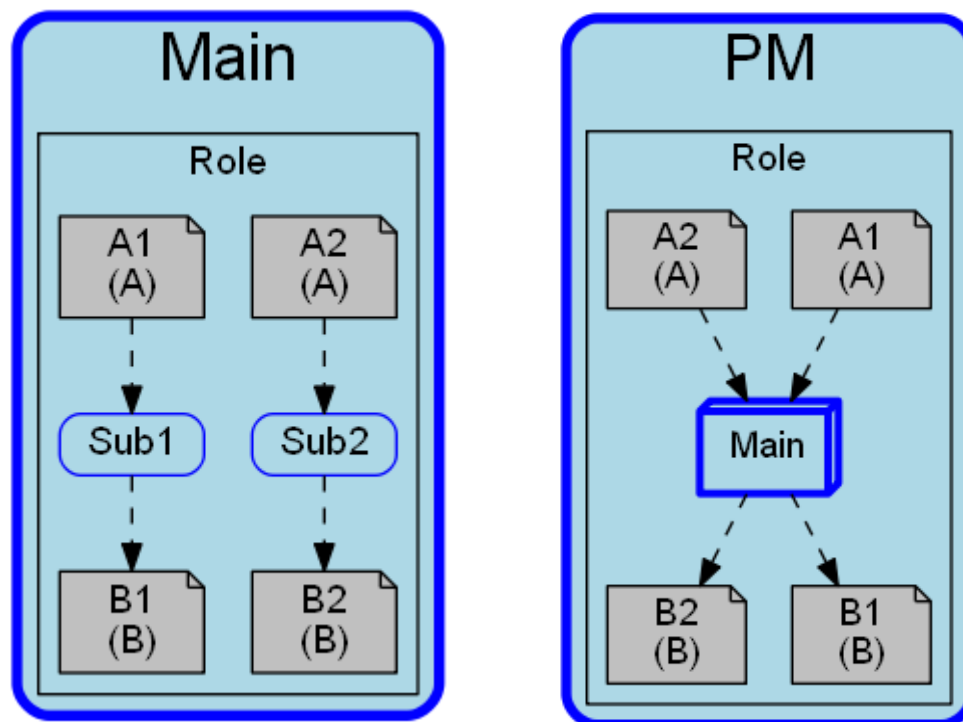


Figure 60: Interface Consistency: Flat Model

The same example can be consistently modeled using hierarchic artifacts. Figure 61 (hierarchic) shows on the left side the same situation as depicted in Figure 60 (flat), except that the process is called “Hierarchic” and used the hierarchic artifact A (contains A1 and A2) as input and hierarchic artifact B (contains B1 and B2) as output, which simplifies the model on the right side significantly.

This hierarchic modeling principle has also the advantage of supporting more abstract processes on higher level and allowing a detailed process

model in lower level by remaining consistent. The example can be found in PMT example directory in the file "InterfaceHierarchy.pmt".

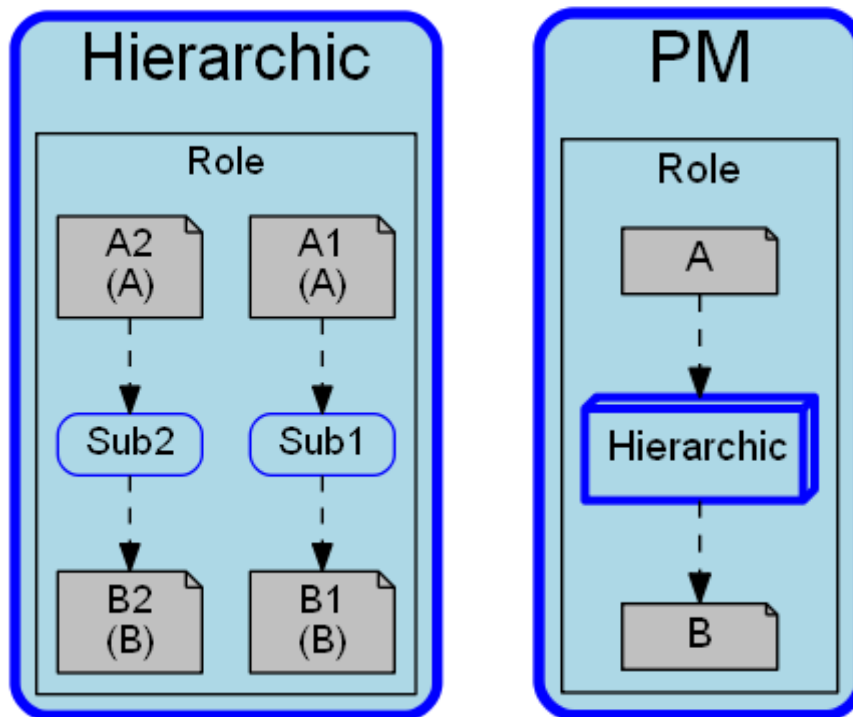


Figure 61: Interface Consistency: Hierarchic Model

8.2.2 Model-Based Processes

Model-based processes can be modeled in the same way as normal processes (see previous section), for example naming an artifact "Model" or "TestModel". However most modeling tools support different modeling styles, by using different modeling elements in different process phases. For example State Charts and Sequence diagrams can be both described within UML. Therefore a detailed process description should differentiate between the different UML models. Furthermore it should be possible to describe the modeling process precisely, for example to specify which modeling elements belong to state charts and sequence diagrams and which not.

All this can be achieved by allowing the process specification to use and refer to the Meta-Model of the used modeling tools. For example in a state chart model state charts and transitions are required and other modeling elements (labels, junctions,..) are optional.

PMT supports specification of models using meta-models with the following modeling elements:

- Model: specialization of Artifact

- MetaModel: Container for Meta-Model elements
- MetaModelElement: Elements of the meta-model, e.g. State, Transition
- MetaModelAttribute: Attributes of meta model elements, e.g. Name, Action, Condition
- MetaModelAssociation: Association between meta model elements, e.g. input-transition

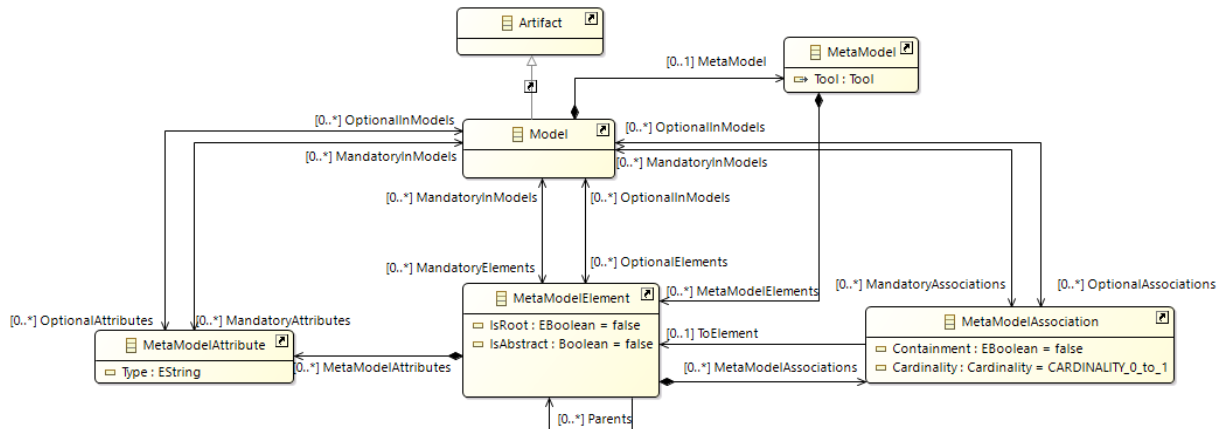


Figure 62: Modeling of Model-Based Processes

The main specification of models are the two relations “Mandatory” and “Optional” from the Model to the different elements of the meta model. Note that ensuring that the models are built according to the specification is not task of the PMT tool, but of the modeling tools. PMT focusses on the specification of the models.

Since creating a detailed meta model can be much work, PMT allows to import existing meta-models from other Eclipse-based tools (“Ecore Import”), see Section 7.2.4.

An example of a model-based artifact is specified in Figure 63. It shows a PMT Model frame that consists of a process element with a name that can contain all elements. In Addition the frame can contain stakeholders and tools, but this is optional.

The image shows a software interface for defining PMT models. It consists of six vertically stacked panels, each with a title bar and three small icons (a square with a circle, a green circle, and an orange circle) on the right. The panels are:

- Mandatory Elements:** Contains one entry: "Meta Model Element metaModel.referenceProcess.Process".
- Optional Elements:** Contains two entries: "Meta Model Element metaModel.referenceProcess.StakeHolder" and "Meta Model Element metaModel.referenceProcess.Tool".
- Mandatory Attributes:** Contains one entry: "Meta Model Attribute metaModel.referenceProcess.Process_name".
- Optional Attributes:** Contains two entries: "Meta Model Attribute metaModel.referenceProcess.StakeHolder_name" and "Meta Model Attribute metaModel.referenceProcess.Tool_name". This panel has a blue dashed border around its title bar.
- Mandatory Associations:** Is currently empty.
- Optional Associations:** Contains two entries: "Meta Model Association metaModel.referenceProcess.Process_stakeHolders" and "Meta Model Association metaModel.referenceProcess.Process_tools".

Figure 63: Model Frame in PMT Models

More details can be found in the meta-model Section.9.12.

8.2.3 Requirements

Requirements are mainly hierarchically. Therefore they are modeled hierarchically as a tree. References to other requirements ("RequiredRequirements") are also supported. Note contained requirements do not need to be modeled as requirements, they are automatically required.

Important is the traceability of the requirements model to the original requirements. This can be modeled using IDs. Figure 64 shows the structure of the requirements in the ModuleTest example.

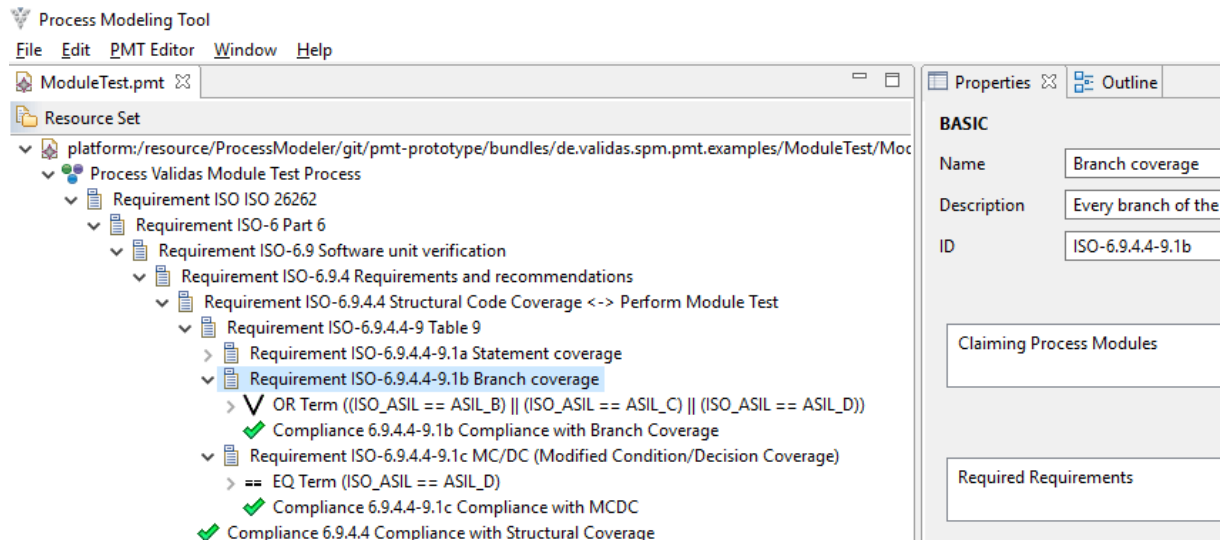


Figure 64: Requirements Example

Note that there are also extended properties that allow to specify the mandatory requirement risk levels, see Figure 65. However for automated tailoring those are not supported. We recommend to use the more general and powerful principle of Variant-Terms that can be automatically evaluated, see the “ORTerms” in Figure 64 for example that describe Variant Terms in a formal way.

Figure 65: Extended Requirement Properties

8.2.4 Compliance

Compliance modeling consists of two parts

- 1) Claiming requirements
- 2) Arguing compliance

Showing/Proving compliance can then be done using the VVT Export.

Claiming requirements describe the process of stating "This process satisfies these requirements". This is not a compliance statement, but rather a compliance goal.

To model claimed requirements, PMT allows to use the two (equivalent) model relations:

- In ProcessModules: Use the association "ClaimedComplianceRequirements" (COMPLIANCE part), see Figure 66
- In Requirements: Use the association "ClaimingProcessModules" (BASIC part), see Figure 67.

Note the PMT tree browser shows the compliance using "<->" after the names of ProcessModules and requirements, see Figure 68.

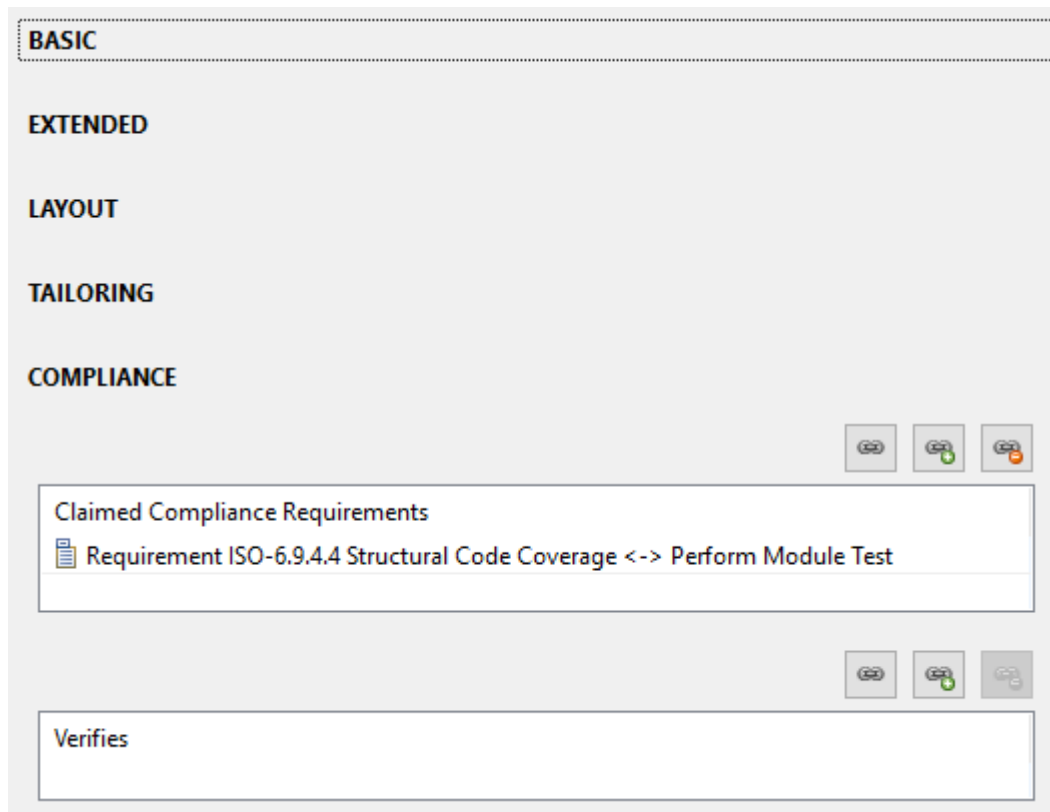


Figure 66: Claimed Compliance Requirements in ProcessModule

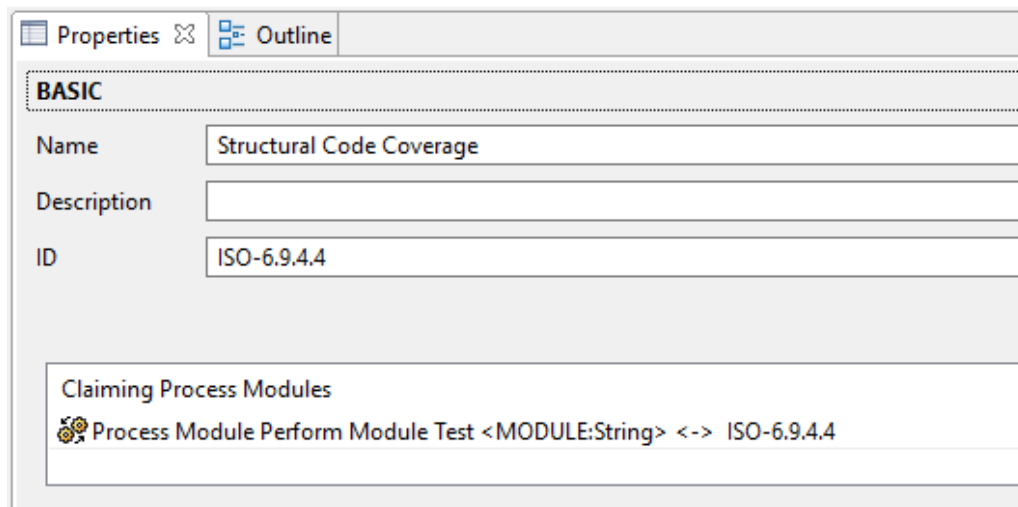


Figure 67: Claiming Process Modules in Requirement

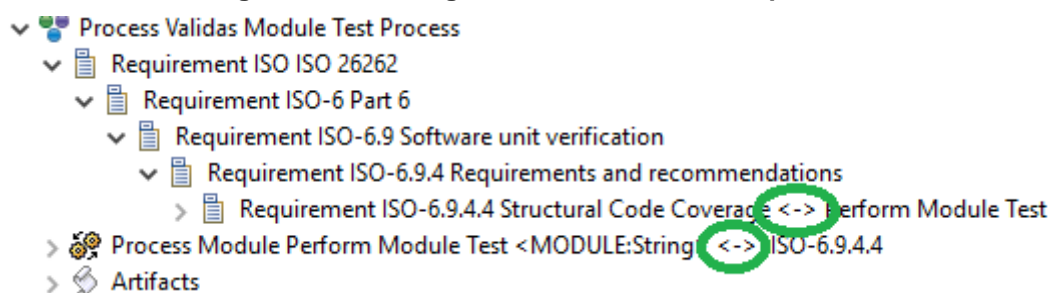


Figure 68: Compliance Claims in Tree-Browser

The compliance argumentation is done by adding Compliance elements. The compliance element provide the linking to the process that are used to satisfy the requirement and to the VerificationModules that are used to verify the correct implementation of the requirements.

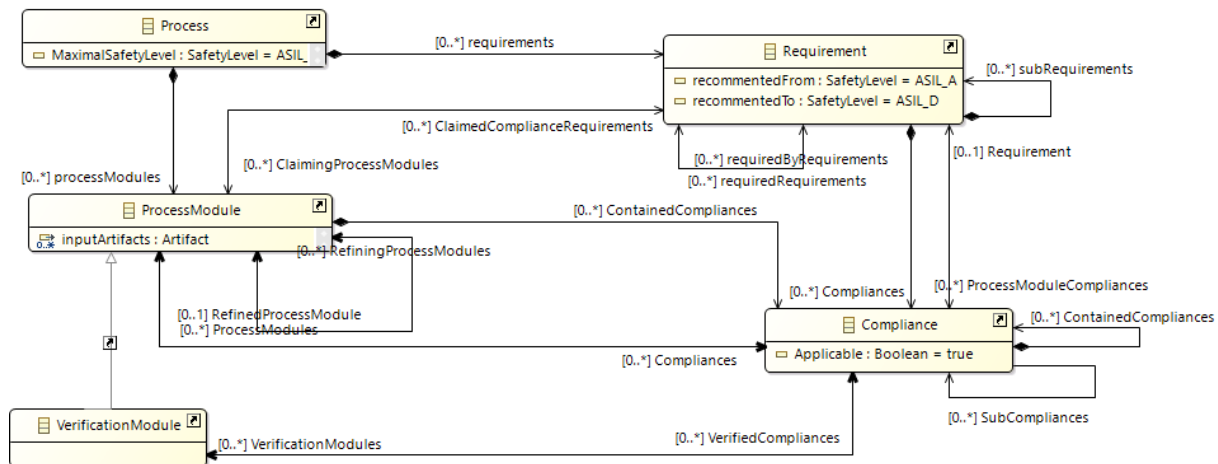


Figure 69: Compliance Elements

The Meta model for compliances shows that there are two containers that can contain Compliance elements:

- **Requirement**: allows to store the compliance elements directly within the requirements
- **ProcessModule**: allows to store the compliance argumentation within the ProcessModule

While the first seems to be more natural from the standard point (it's easier to manually check for completeness), the second is more modular, especially if ProcessModules are re-used and refined.

If a Compliance element is stored within a Requirement element it demonstrates the compliance to it's containing element. If a requirement is contained within a ProcessModule the Requirement has to be explicitly set (using the "Requirement" association).

If a ComplianceElement is stored within a ProcessModule (or VerificationModule) the satisfying Process is the container. If it is stored within a Requirement, the ProcessModule and VerificationModule elements have to be linked explicitly.

For hierarchic requirements the compliance argumentation is usually also hierarchic. If stored within the requirements (or ProcessModules), the "SubCompliances" have to be linked in order to close the argumentation. If Compliance elements are stored within other compliance elements as "ContainedCompliances") there is no need to set the SubCompliance link.

The best way to create compliance argumentation is to

- Select a requirement (usually the main goal, i.e. a hierarchic requirement)
- Generate a compliance structure with PMT (including the links to the requirements)
- Move this compliance structure into the process that is compliant
- Fill the compliance argumentation with arguments, process links and V&V activities.

The second step can be done using PMT action **Generate Compliance Structure**, that is in the popup menu of the requirement action. The action creates a compliance argumentation tree with

- The same hierarchy as the requirements
- Compliance argumentations for hierarchic requirements using a default argumentation that the requirement is satisfied since all sub-requirements are satisfied.
- Variant terms in the compliance argumentation, provided that the requirements had variant terms

8.2.5 Reuse & Linking

Processes can be reused in several ways:

- 1) Just reuse them as they are.
- 2) Instantiate them several times, see Section 8.2.7 for instantiation of processes.
- 3) Reused within modeling.

The last point is scope of this section.

In general the model is a tree, visualized in the tree-browser. However it is also possible to reuse elements by adding references. The following elements support re-use:

- ProcessModules: process modules can refer to other process-modules using the relation "SubProcessModuleReferences".
- ProcessModules: can also re-use Parameters using the relation "ParameterReferences".
- Artifacts: Artifacts can refer to other artifacts using the relation "SubArtifactReferences".

- Models: Models can refer to other models using the relation "Includes".
- Requirements: Requirements can refer to other requirements using the relation "RequiredRequirements"
- Terms: Terms can re-use enumerated constants using the modeling element "EnumValueRef" and terms can refer to Parameter values using the model element "ParamRef".

References are not visible in the tree-browser, but in the generated documents and the other views in PMT: Properties, ProcessView, ComplianceView.

If process modules shall be re-used it is recommended to create a process library and re-use processes from that in the modeled processes. See Figure 70 for an example of reuse.

Figure 70 shows a Library with a Verification and Validation Process that is included (using "SubProcessModuleReferences") in all three qualification kit processes of Validas. The model also contains a switch "Project:ProjectKind" i.e. a ProcessVariable with an enumerated type that allows to select a current project kind by binding the value. In the example the switch is bound to the enumerated value "LibraryQKit", which enables the library process and disables all other projects by tailoring. The process view of the process "Library QKit" shows the included/referred sub-process "V&V" which is not visible in the tree-browser, since it is included as a reference into the process.

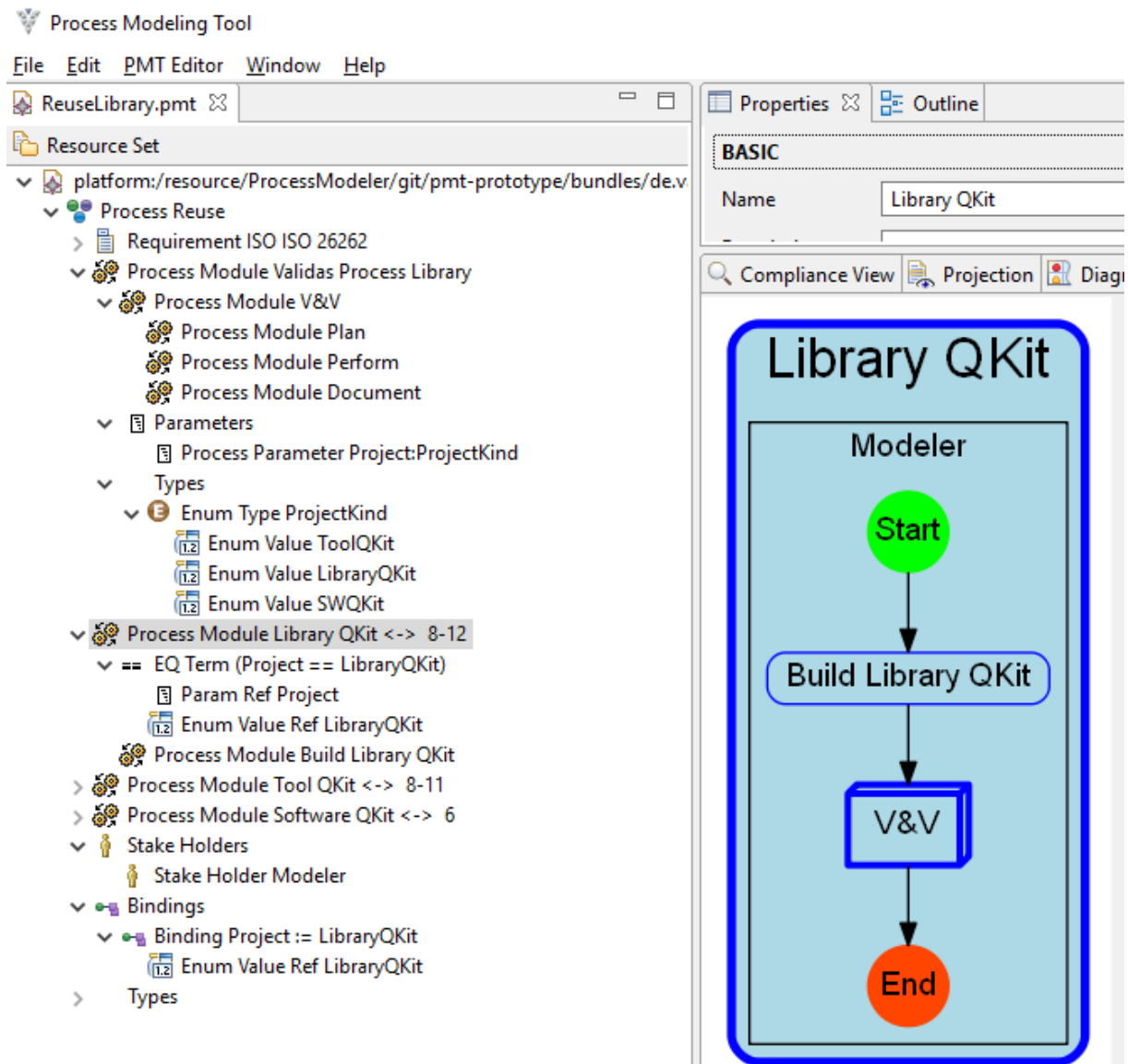


Figure 70: Reuse Example

8.2.6 Tailoring

Tailoring describes the adaptation of a generic process to a specific project.

In PMT this is done automatically by using so called "Variant Terms", i.e. terms over ProcessVariables that describe the condition under which the process is present. If the term evaluates to false the corresponding modeling elements are "tailored away". If the condition evaluates to true the corresponding element will be considered. Note: If the term cannot be evaluated then the element is also present. This is true in Generic processes that can be tailored.

The description of the evaluation is contained at the end of Section 8.1 (Term Evaluation). The parameters are described in Section 8.2.8. All used modeling elements (Terms, Parameters) are described in the meta-model Section 9.

The re-use example of the previous section (Figure 70) can be used to illustrate the tailoring. The tailoring can be analyzed by selecting the project and starting the action "Evaluate Variants" which results into the following information, see Figure 71.

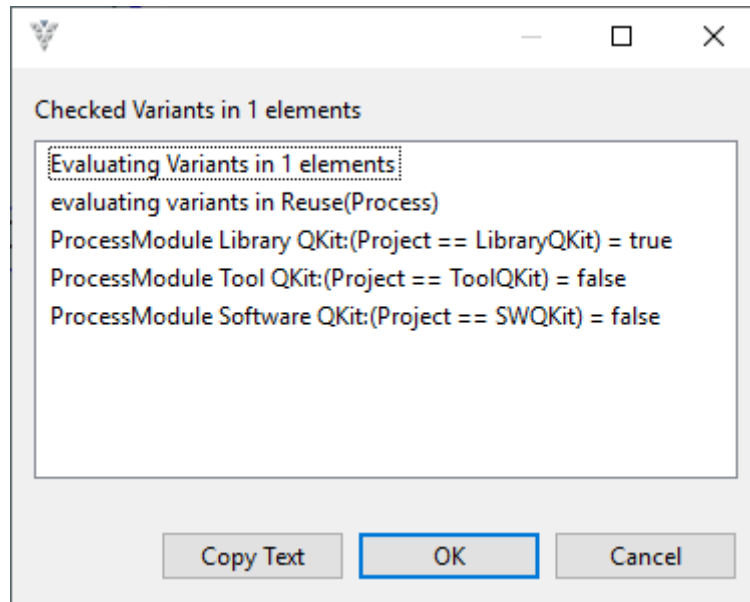


Figure 71: Evaluate Variants Result

8.2.7 Instantiation

Typically processes need to be instantiated several times (unless this is not only done during V&V in VVT), this can also be done within PMT. There are several modeling steps and properties that can be used to create new process modules and to mark them as "Instances" of the generic process (see Section 9.10.1).

Of course Instantiation of processes makes only sense for parameterized processes. Typically the parameters are fixed within the project.

However there is a simplified way to create Instances of processes. This can be done by the following steps:

- Creation of an additional ProjectParameter "LIST_OF_<Parameter>" that contains all values that shall be instantiated to the parameter.
- Link the LIST_OF-Parameter to the process parameter as "Values from" or "Iterator" parameter, such that the process parameter iterates over the list parameter, i.e. received the values from it, see Figure 72. Note that the list parameter has to have a List-Type, usually list of String.
- Create a Binding with all values of the list parameter (as a list term) referring to the list variable and having the list-term contained.

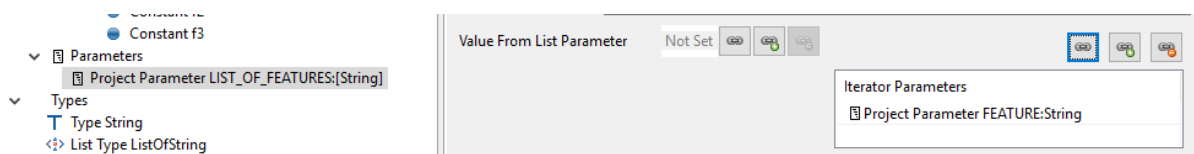


Figure 72: Assignment of Iterator Parameters

An example for a generic process, ready for instantiation as described above is depicted in Figure 73. It shows the list parameter, as well as the parameter definition in ProcessModule Specify Test Cases. Other module (Implement Test Cases and Verify Tests) refer to the parameter.

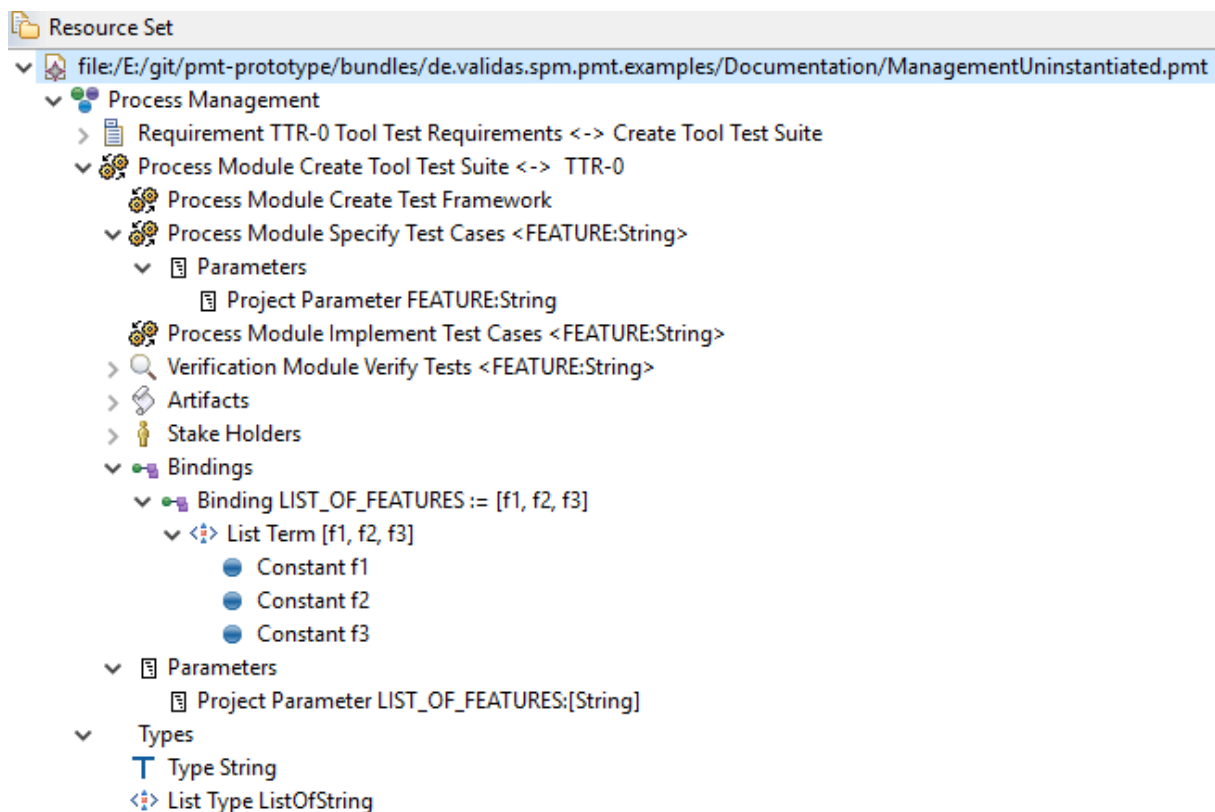


Figure 73: Example for Automatic Instantiation

On a well-defined project the instantiation can be started by selecting the process to be instantiated and by selecting the popup action "Instantiate Process Module", see Figure 74.

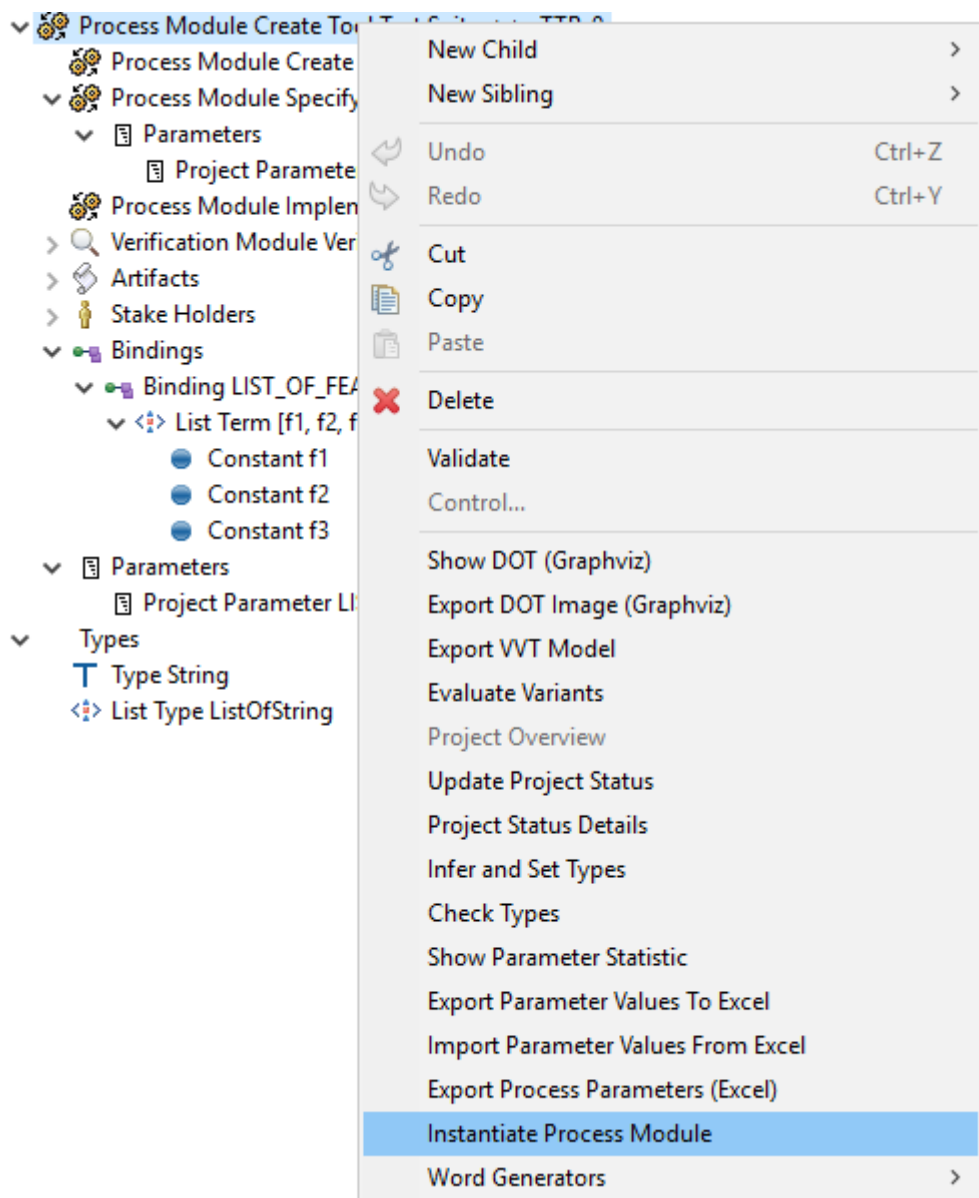


Figure 74: Starting Automatic Instantiation on Process Module

After starting the action, the user is asked to confirm the instantiation as shown in Figure 75.

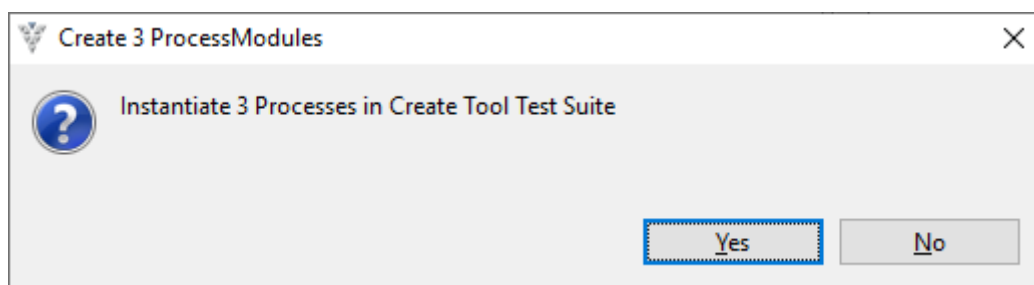


Figure 75: Confirmation of Automatic Instantiation

Finally PMT will display an information about the instantiated process modules as shown in Figure 76. Note that only processes which have

parameters or references to them) will be instantiated, others not (Create Test Framework in the example).

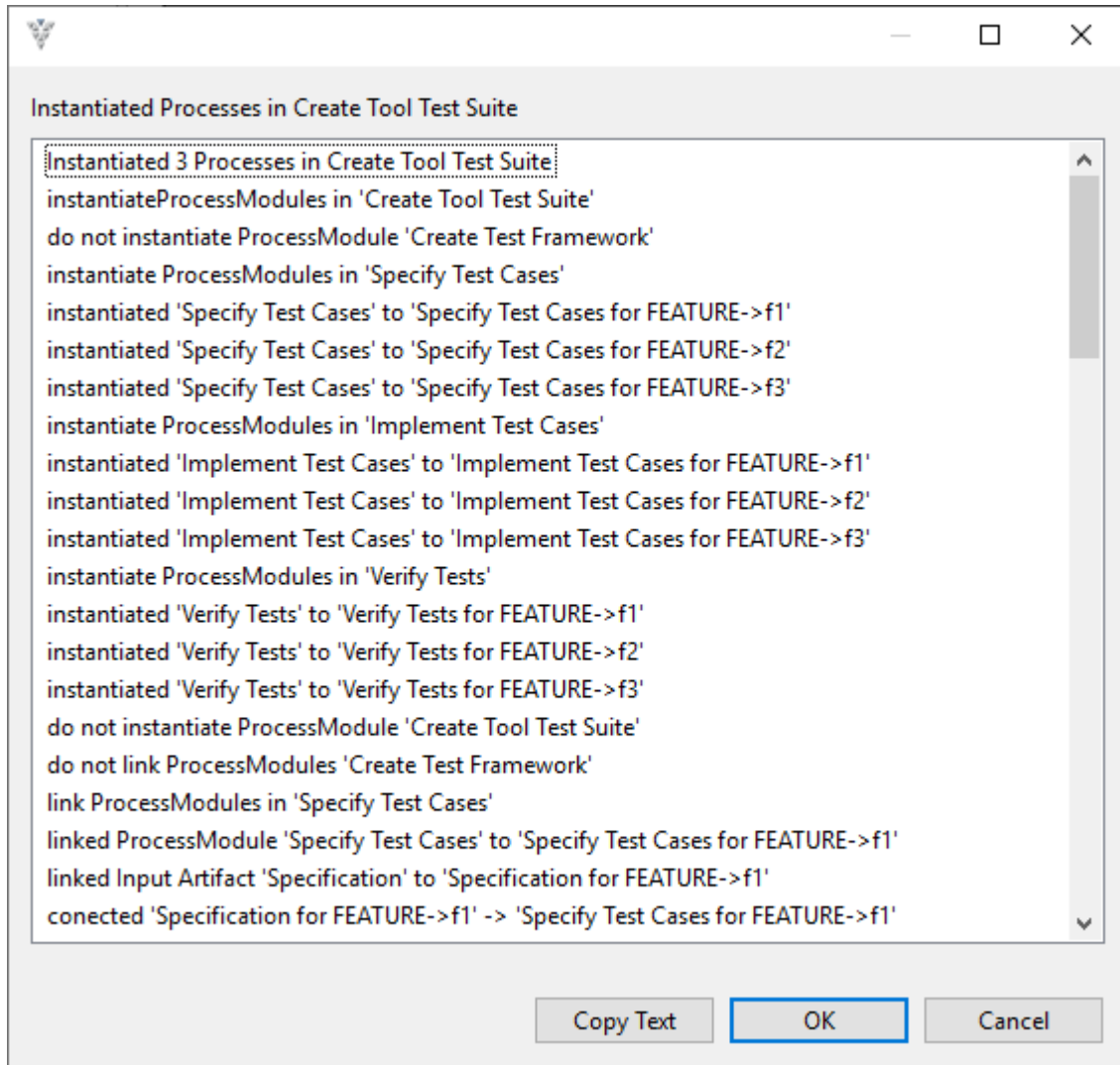


Figure 76: Result Information of Automatic Instantiation

The result of the instantiation is that the parameterized processes have their instances as sub-processes, see Figure 77 and Figure 78. Note that not only the processes have been instantiated, but also the artifacts have "Sub-Artifacts" for each instance: The artifact "Test Specification" has now three sub-artifacts:

- Test Specification for FEATURE->f1
- Test Specification for FEATURE->f2
- Test Specification for FEATURE->f3

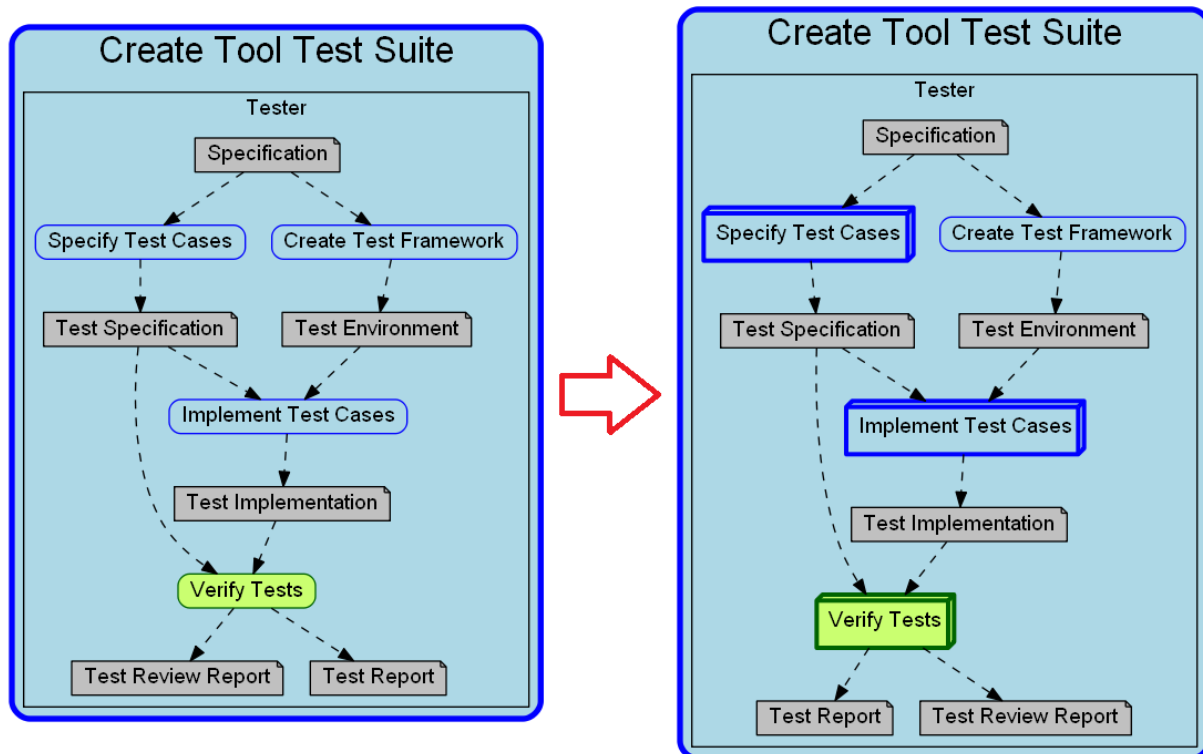


Figure 77: ProcessModule Instantiation

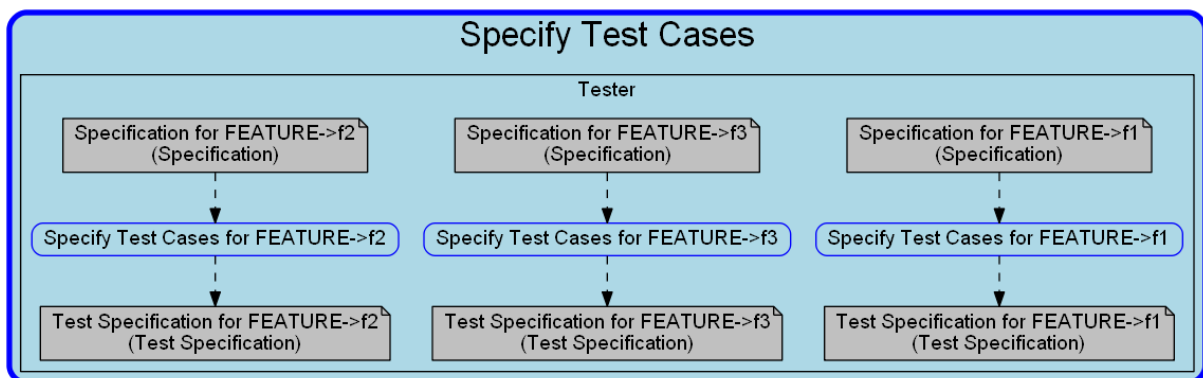


Figure 78: ProcessModule Created Instances

8.2.8 Variables & Variants

Variables and Variants are used for tailoring, see Section 8.2.6.

8.2.9 Layouting

The graphical notation of the processes is done automatically by converting the model into the dot format of graphviz, see <http://www.graphviz.org> for more details.

The representation can also be seen (for debugging) by exporting it into textual format and piping it into an online tool as graphviz for example.

The user can impact the layout in two ways:

- 1) By specifying layout priorities in the elements.
- 2) By specifying "invisible" between elements using "layout before/after" properties

Consider the example in Figure 79. It contains a process module "PM" with three sub-processes: "A", "B" and "C". The order of them is not determined, since they are independent, so the left from right order is undetermined by the model. In the example in Figure 79 the order is "B" before "C" before "A".

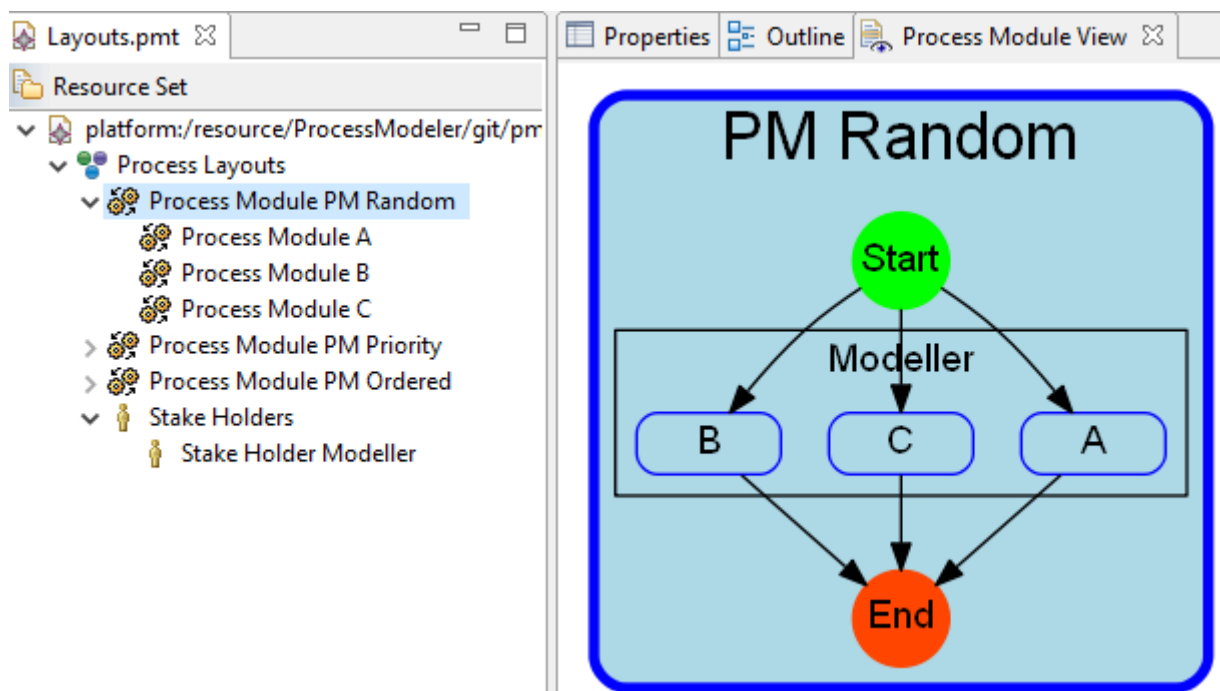


Figure 79: Layout Example: Unlaid (random)

If the user wants to change the order to "A" before "B" before "C" it can be done using the Layout priority. Setting the LayoutPriority attribute to

- A=30
- B=20
- C=10

This generates the desired layout as shown in Figure 80:

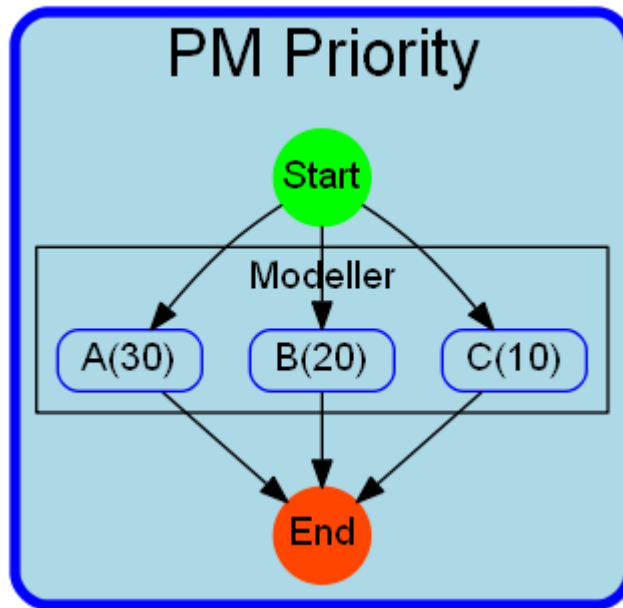


Figure 80: Layout Example: Laid out using Priorities

If the elements shall be ordered then the layout will be done by adding (“invisible”) lines as specified. Figure 82 shows the result by specifying the ordering using “LayoutBefore” and “LayoutAfter”. Please note that from the view-point of “B” (middle), B is layouted after A and before C. So hence the attributes in B are specified as shown in Figure 81.

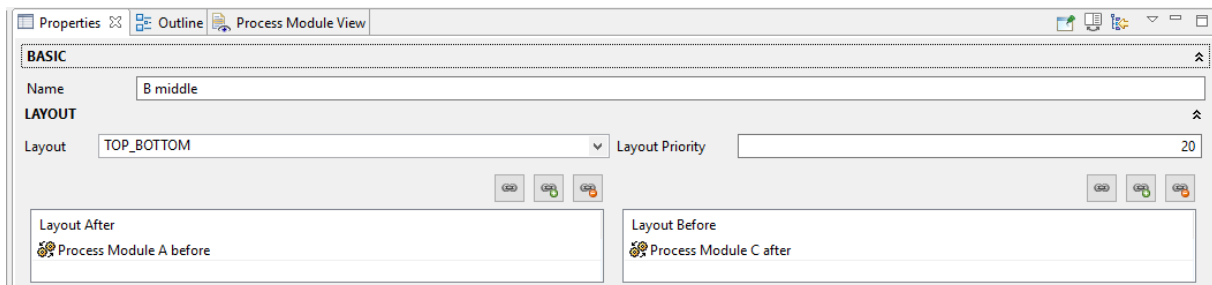


Figure 81: Layout Example: Specification of Layout Order

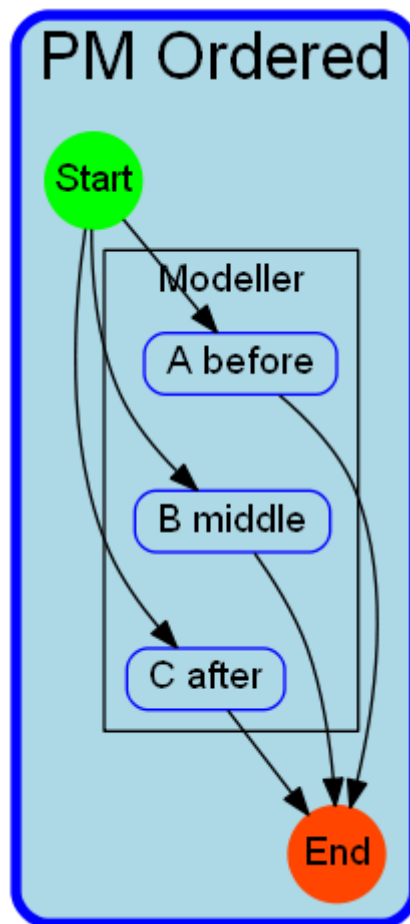


Figure 82: Layout Example: Laid out by Ordering

8.2.10 Tools

Tools can be used to support the application of methods in processes. In safety relevant processes it is important to use the tools safely. Therefore they have to be classified and eventually qualified. The TCA tool, see <http://www.validas.de/en/services/tca/> is a special purpose tool for all aspects of tool qualification. Therefore we focus in PMT only of the safety process relevant aspects. These are

- The use cases of the tool (within the process)
- The supported methods required from safety standards
- A preliminary classification with an explaining argument.

Note that the preliminary classification is not standard compliant but might be an indicator for the selection of tool candidates that can be qualified, or do not need to be qualified.

Tools are modeled globally in the Process container as shown in the example in Figure 83.

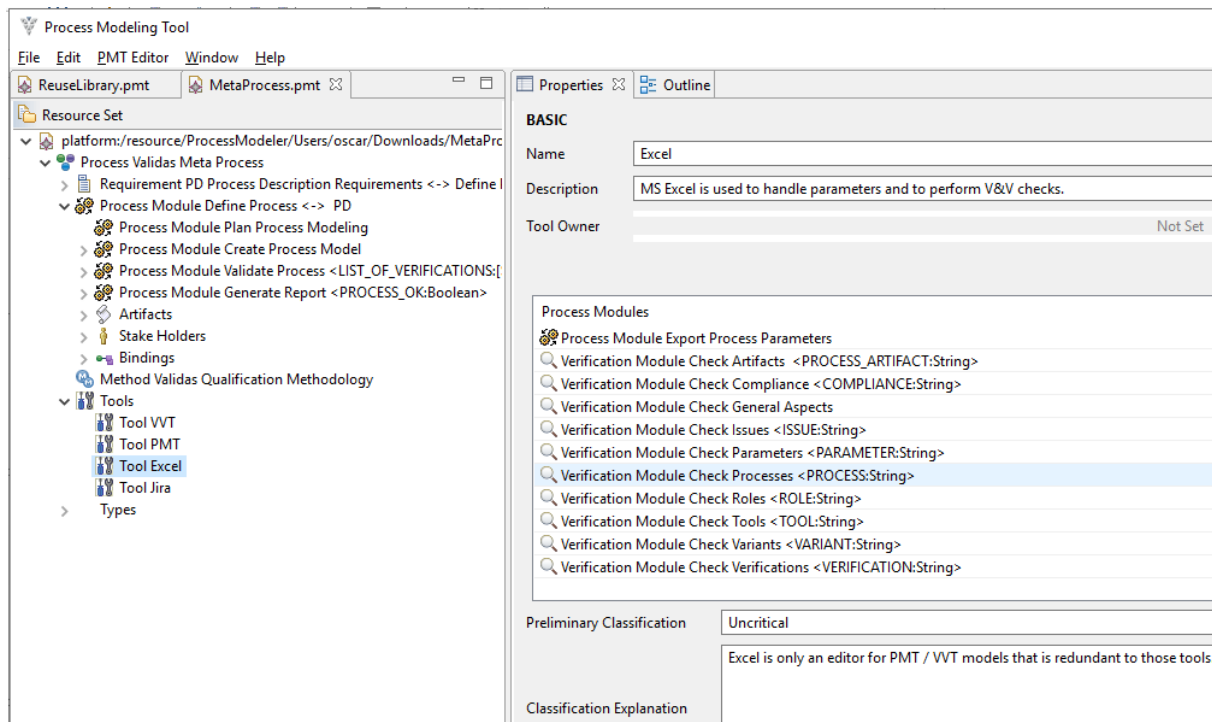


Figure 83: Tool Modeling

8.2.11 Project Management

It is not the goal of PMT to replace professional project management tools. PMT can provide input to them and can be used to demonstrate some requirements on project management tools by some simple features, like management of instances or requirements coverage. PMT supports project management by some basic features

- Checking requirements coverage (useful during process development)
- Determining efforts
- Managing instances of processes

The Project Management Informations are grouped within the collapsible group "Management" in the property view.

The PMT manages the status of the projects by

- A status of the artifacts with the following values (see Section 9.2.1):
 - DEFINED
 - PLANNED
 - READY
 - IN_PROGRESS
 - DONE
- The same status can be specified for the ProcessModules

Note that PMT can determine the status semi-automatically, i.e. if all inputs of a task (=ProcessModule) have the status DONE, then PMT can infer the status of the task to "READY". If user manually sets the status of the tasks to DONE (or IN_PROGRESS), PMT can update the status of the tasks outputs accordingly. The PMT status updates can be triggered using the action "Update Project Status" on Process and ProcessModule elements.

8.2.11.1 Project Overview

PMT can compute the project overview by starting the Action "Project Overview" on Processes (see 7.2.2).

This results into a textual description of the project status including the following information (see Figure 84):

- Effort Computation (and completeness): In case there are no efforts specified (value=0) status will contain warnings: "Could not compute effort for", see Figure 84.
- Number of Requirements to satisfy
- Number of Compliances to contribute, including percentage of compliance
- Numbers of Required Process Modules (to achieve compliance) and Verification Modules
- Status of Tasks (ProcessModules) and Artifacts

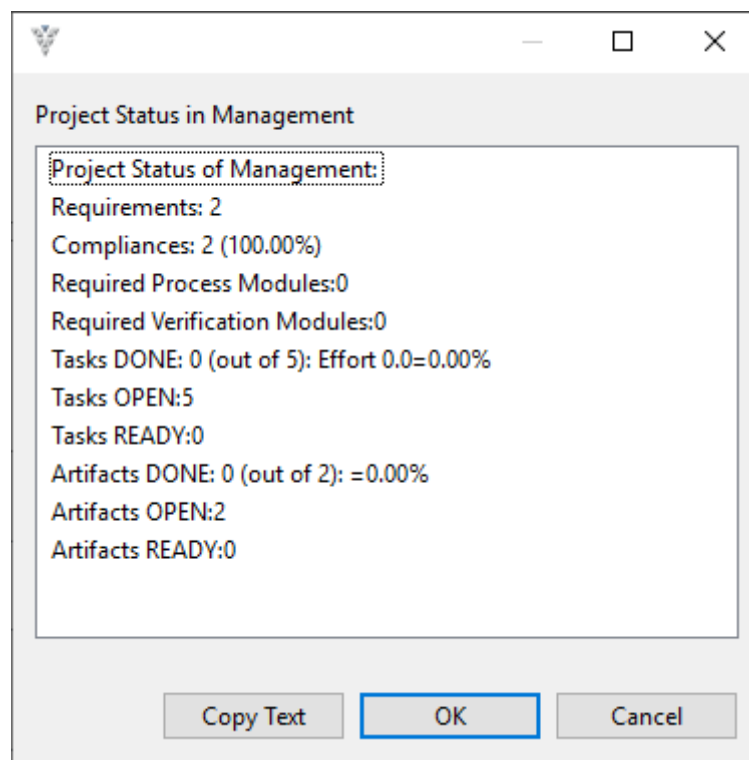


Figure 84: Initial Project Status (Overview)

8.2.11.2 Project Status Details

PMT can compute the status details on projects. This is based on the status of the artifacts and on the status of available inputs.

This results into a textual description of the project status including the following information (see Figure 84):

- Effort Computation (and completeness): In case there are no efforts specified (value=0) status will contain warnings: "Could not compute effort for", see Figure 84.
- Number of Requirements to satisfy
- Number of Compliances to contribute, including percentage of compliance
- Numbers of Required Process Modules (to achieve compliance) and Verification Modules
- Status of Tasks (ProcessModules) and Artifacts

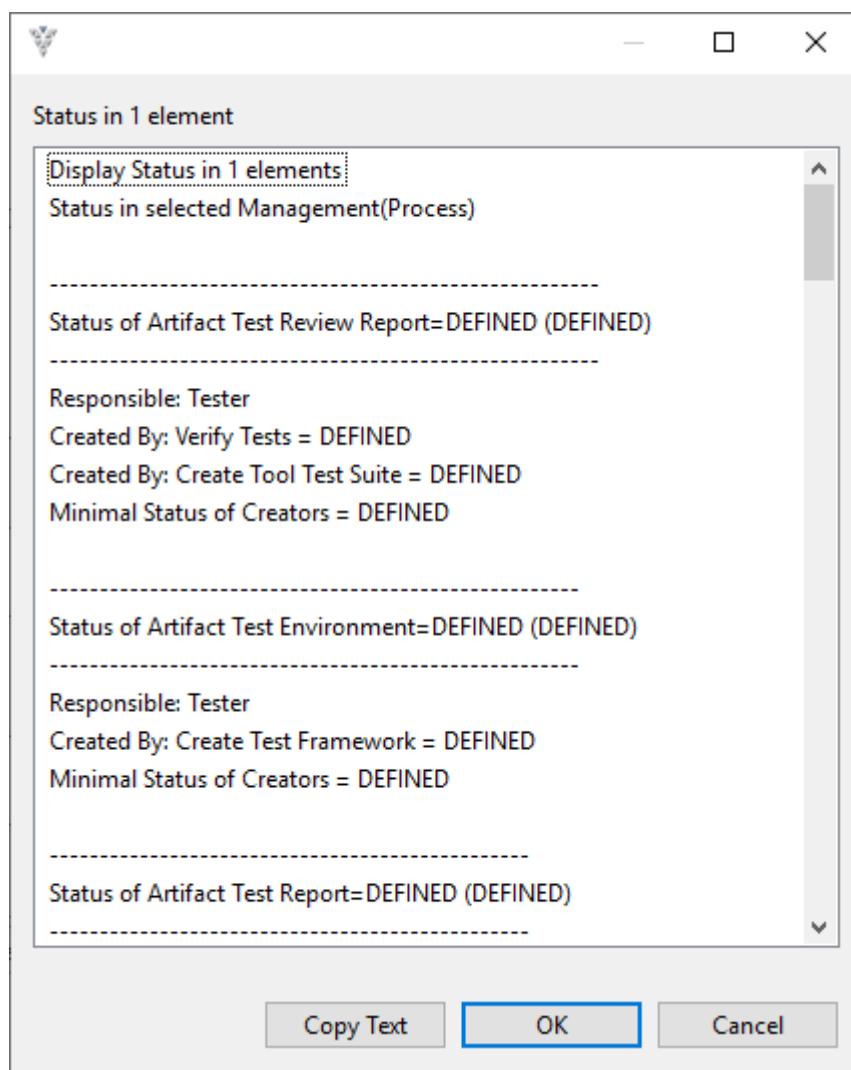


Figure 85: Initial Project Status (Details)

8.2.11.3 Excel-Interface for Project Management

To interface with other project management tools, PMT uses an Excel-interface to export & import the project status. Export & import can be started from ProcessModules using the right mouse button (popup-action) as shown in Figure 86.

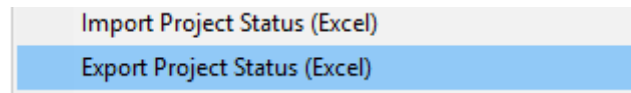


Figure 86: Excel-Interface for Project Status

The Excel Interface consists of the following Information (see Figure 87) for all manageable events:

- The type: Artifact/ProcessModule/VerificationModule
- The name
- The state
- The ID (if specified)
- The qualified name (separated using ".", which should not occur in names)
- Description
- Effort (only for ProcessModules & VerificationModules)
- Progress (only for ProcessModules & VerificationModules)
- Planned Start Date (only for ProcessModules & VerificationModules)
- Planned End Date (only for ProcessModules & VerificationModules)

	A	B	C	D	E	F	G	H	I	J
		Name	State	ID	Qualified Name	Description	Effort	Progress	Planned Start	Planned End
1	Type									
24	Artifact	Test Specification for FEATURE->I2	DEFINED		Create Tool Test Suite.Test Specification.Test Specification for FEATURE->I2	Concrete s				
25	Artifact	Test Specification for FEATURE->I3	DEFINED		Create Tool Test Suite.Test Specification.Test Specification for FEATURE->I3	Concrete s				
26	ProcessModule	Create Tool Test Suite	DEFINED		Create Tool Test Suite	Creates a	0	0	25.04.2019	25.05.2019
27	ProcessModule	Create Test Framework	DEFINED		Create Tool Test Suite.Create Test Framework	Create a te	40	0		
28	ProcessModule	Implement Test Cases	DEFINED		Create Tool Test Suite.Implement Test Cases	implement	4	0		
29	ProcessModule	Implement Test Cases for FEATURE->I1	DEFINED		Create Tool Test Suite.Implement Test Cases.Implement Test Cases for FEATURE->I1	implement	5	0.01		
30	ProcessModule	Implement Test Cases for FEATURE->I2	DEFINED		Create Tool Test Suite.Implement Test Cases.Implement Test Cases for FEATURE->I2	implement	4	0		
31	ProcessModule	Implement Test Cases for FEATURE->I3	DEFINED		Create Tool Test Suite.Implement Test Cases.Implement Test Cases for FEATURE->I3	implement	4	0		

Figure 87: Project Status in Excel

The status can be changed within Excel and will be-reimported into PMT. All changes will be listed in an import log message, see Figure 88 for an example. Please check this log carefully for errors and warnings. Note that the updated model will not be marked as changed in PMT (due to a current known issue) and save it manually to a file.

Note that the "qualified name" is used to find & identify the elements in the model. All other elements can be changed, for example also the name or the ID.

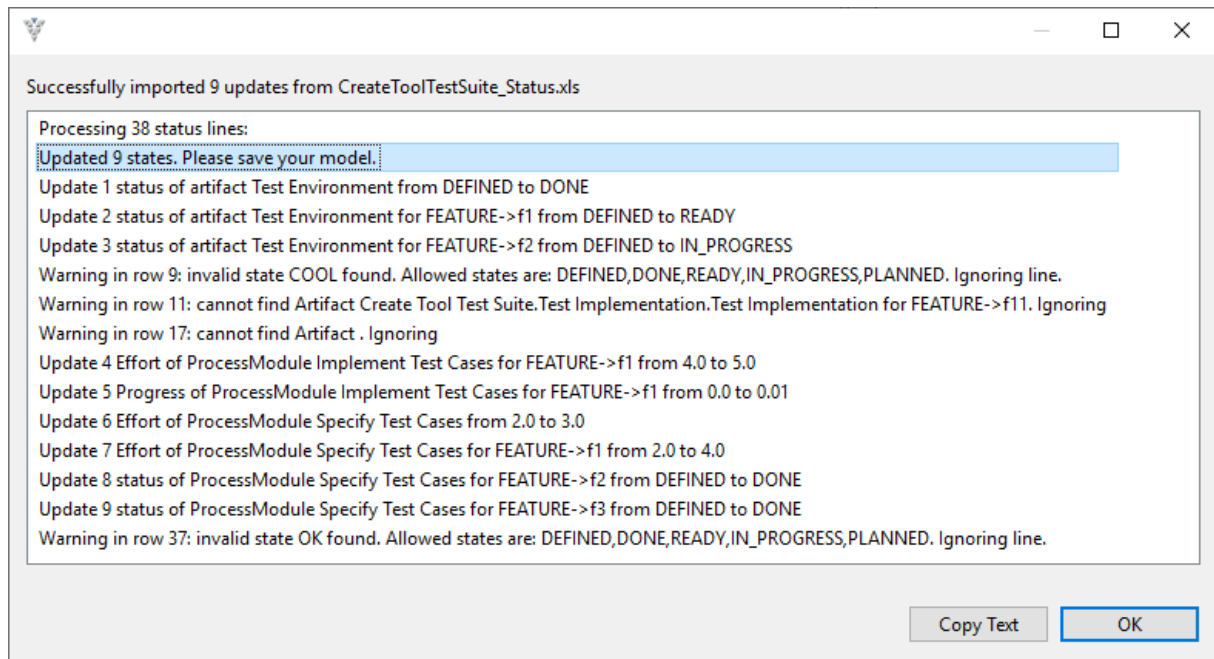


Figure 88: Project Status Import Log

8.2.11.4 Project Status Update

Based on a current project status PMT can compute the next tasks. For Example if all inputs and preceding tasks are in status DONE, than the task can be set to status READY. This status update can be triggered using the following action "Update Project Status" (on Process- and ProcessModule elements), see Figure 89.

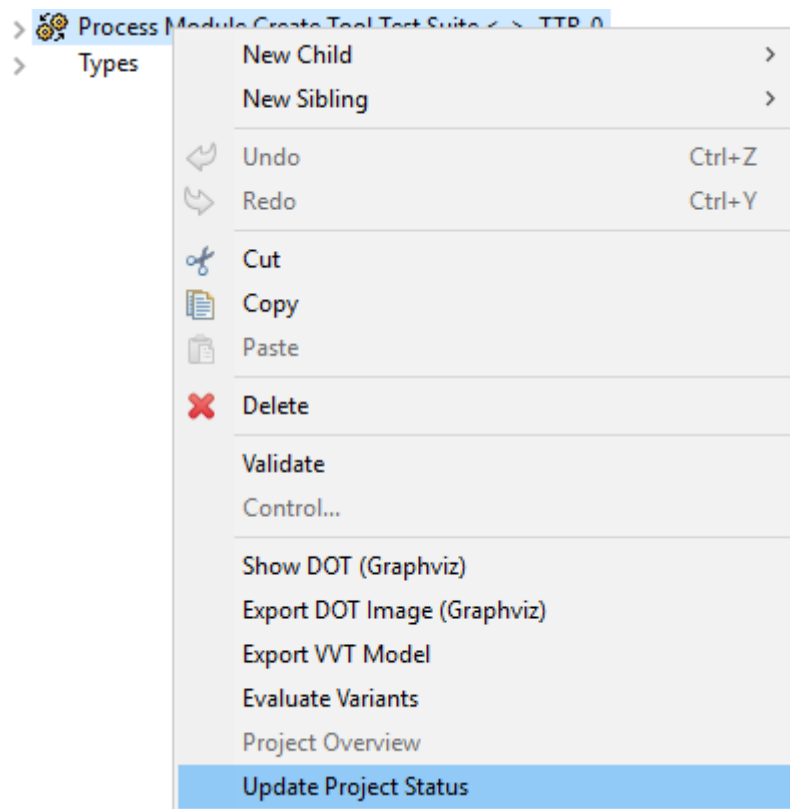


Figure 89: Start of Status Update

The result (changes states) is show in a text dialog, see Figure 90.

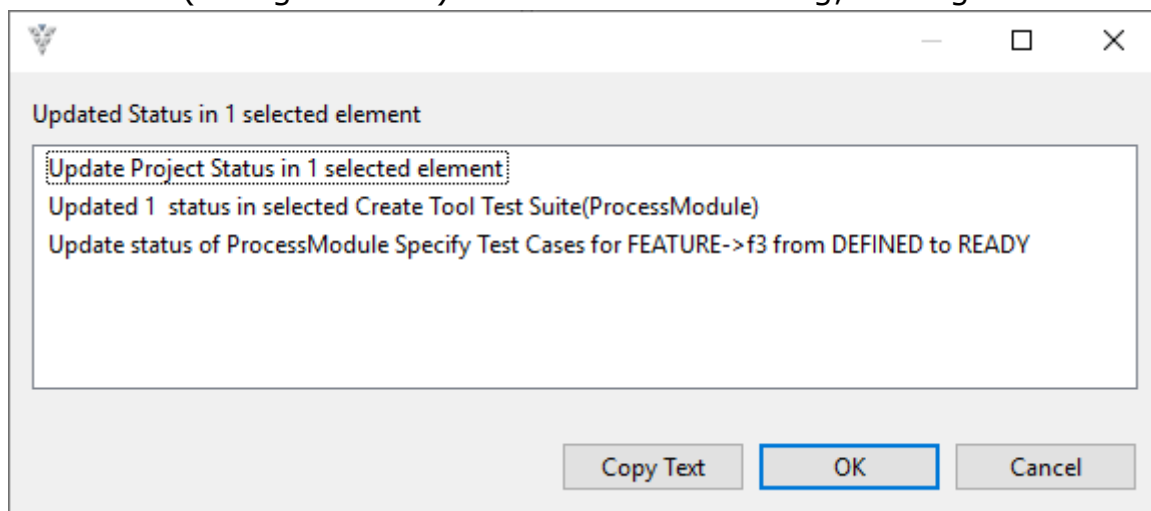


Figure 90: Result of Status Update

8.3 Consistency & Process Interfaces

A process is considered to be consistent if the inputs and outputs of it are consistent with the sub-processes (if available). For example if a process P with only one sub-process S has an input the input must also be an input

of the sub-process. Otherwise there would be “unused” inputs in the process. If many sub-processes are contained, than at least on process should need the input and should process it.

While this is an obvious consistency condition it is hard to enforce due to two reasons

1. It has to be (automatically) checkable
2. It might lead to huge number of input outputs on the top-level and contradict the desired level of abstraction.

The first item is solved by adding a validation rule to PMT: “Process modules have compatible interfaces”. The second is achieved by using the recursive structure of artifacts. On the containing process it is allowed to have composed artifacts (hierarchic) as inputs and outputs. The sub-processes can then access the sub-elements of the artifacts.

Figure 92 and Figure 93 show an example of a main process “Main” with two Sub-Processes “Sub1” and “Sub2” that process hierarchic artifacts “A” and “B” as described in the tree browser depicted in Figure 91. The abstract view (Figure 92) of the process module “Main” shows the input and output using the composed artifacts A and B. The detailed, inner view of “Main”, see Figure 93 shows how the sub-components work on the sub-artifacts.



Figure 91: Hierarchic Artifacts for Interface Example

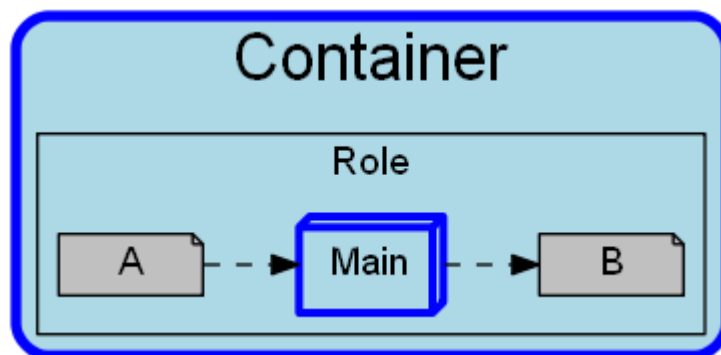


Figure 92: Abstract Interface (Upper-Level)

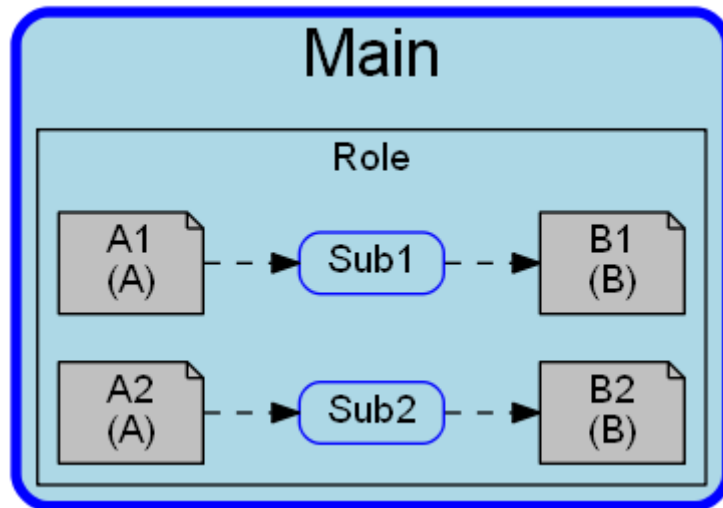


Figure 93: Specific Interface (Lower-Level)

8.4 Refinement

Refinement is a frequently used concept for handling similar but different things, for example in Object oriented programming languages, where “inheritance” is useful.

PMT supports also refinement of processes. A general process can be refined by more specific processes, for example a compiler qualification project might specialize a general qualification project by adding additional inputs (Language compliance suite) or additional outputs (coverage report of the compiler). Also it is possible to define a more specific artifact, e.g. a compiler qualification report that refines the tool qualification report.

If a process refines another process the input output artifacts of the refining process are the addition of general artifacts and the specific artifacts, except if the specific artifacts refine the more general ones.

The refinements can be specified within the Extended property tab of ProcessModules/VerificationModules and Artifacts/Models. Refinements are visualized as displayed using “:” in the tree browser, see Figure 94.

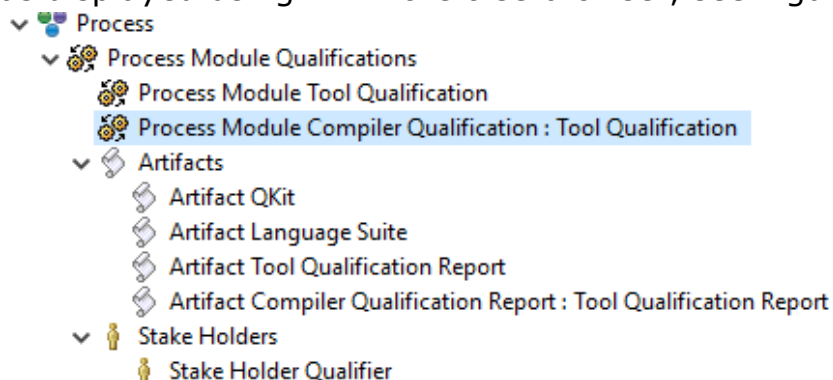


Figure 94: Structure with Refinements: Compiler Qualification refined Tool Qualification

The graphic representation of the refinements is done using “(refines <name>)” annotations, see Figure 95.

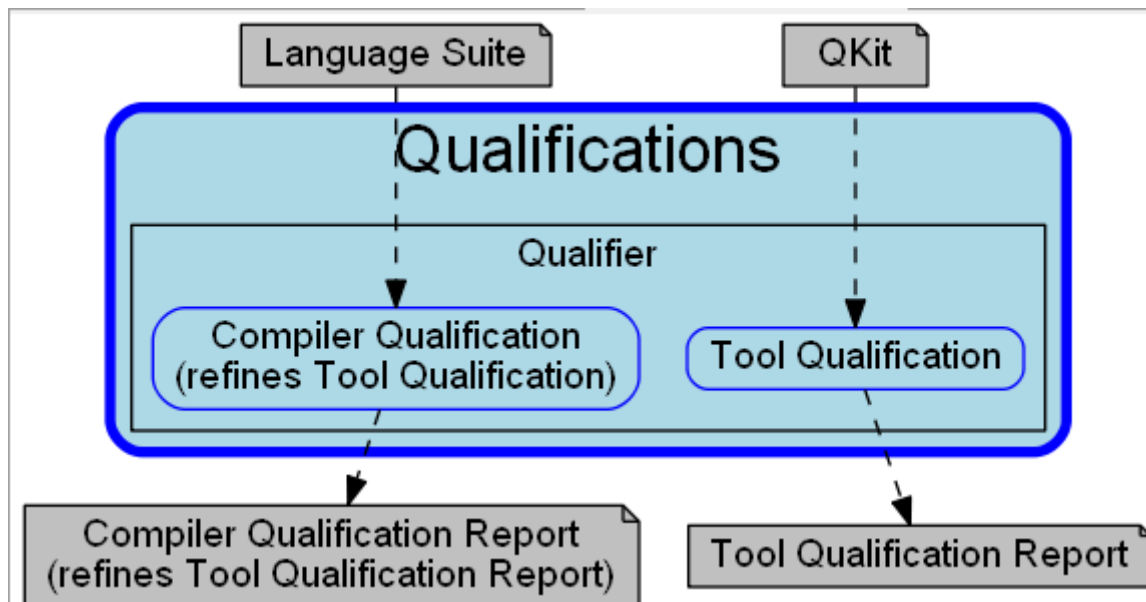
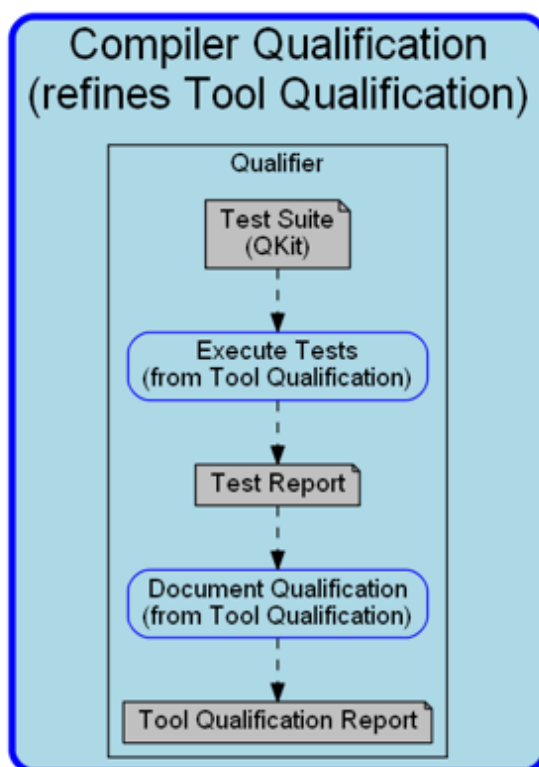


Figure 95: Graphical Notation of Refinements

The “inherited” artifacts can be seen in the process report (and compliance report) that lists all artifacts (alphabetically) including annotations for inherited artifacts in the lists of Inputs and outputs, see Figure 96. Note that the diagrams show the inner view representing the contained sub-processes and their input outputs, hence the Input “Language Test Suite” is only visible in the outer view of the containing processes.

ProcessModule: Compiler Qualification

View of Compiler Qualification



Name:

Compiler Qualification

Description:

A compiler qualification is a frequent case where the qualified tool is a compiler. Usually test strategies for compilers are hard to define, therefore code coverage (within the compiler) might be used to create the confidence.

Qualified Name:

Compiler Qualification

Refines Process Module:

Tool Qualification

Owner (Inherited):

Qualifier

Inputs:

- Language Suite, see Table 27
- QKit from refined Tool Qualification, see Table 28

Outputs:

- 'Compiler Qualification Report' refines 'Tool Qualification Report' from Module 'Tool Qualification', see Table 24
- Language Suite, see Table 27

Figure 96: Generated Table for Compiler Qualification

For the sub-processes of refining elements the same principle applies as for the interfaces: if a process module refines another process module it automatically inherits all sub-elements from the refined elements. If a

Sub-ProcessModule refines an inherited sub-ProcessModule it “overwrites” / replaces it, i.e. the refined element is not inherited.

This is illustrated using the example refinements that shows different qualification processes. Figure 97 shows an example (simplified) qualification process with two sub-processes: “Execute Tests” and “Document Qualification”.

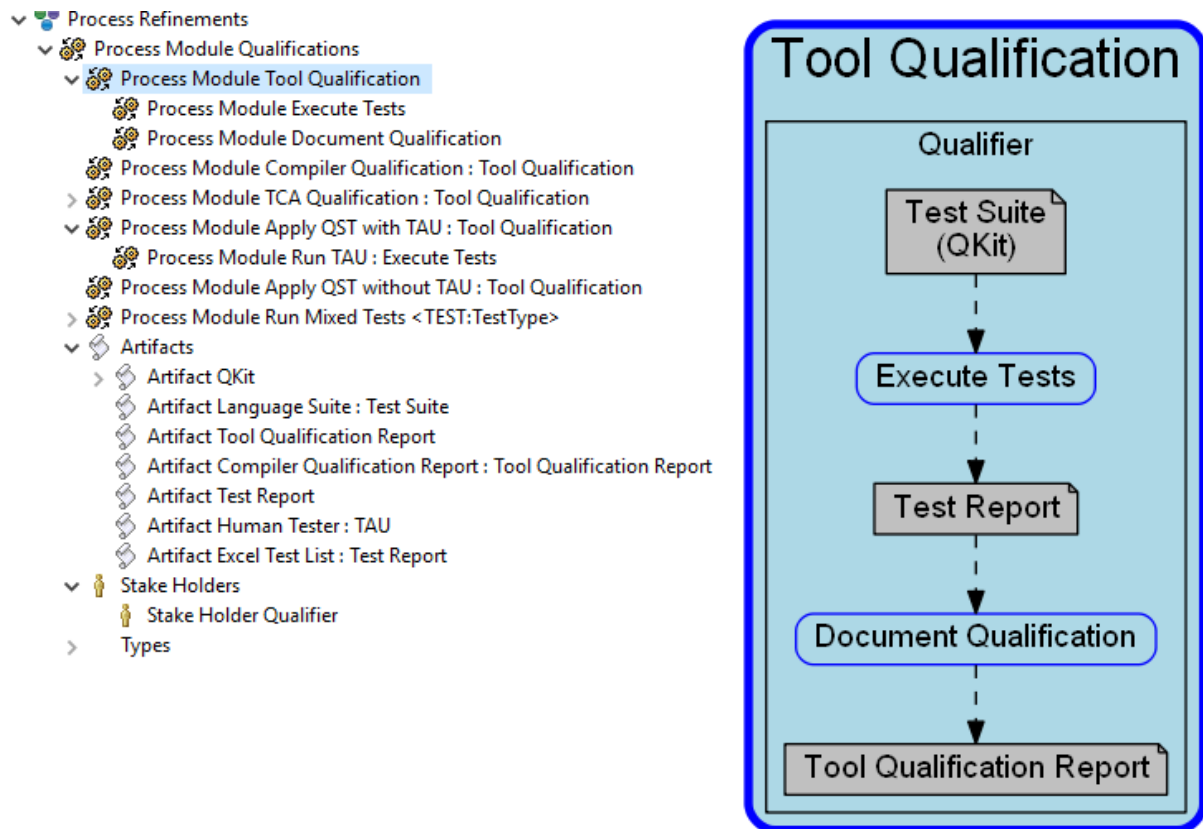


Figure 97: Simplified Qualification Process

The Process Module “Apply QST with TAU” refines this module and “replaces the “Execute Tests” with the automated “Run TAU” and replaces the “Document Qualification” by “Generate Documentation”, see Figure 98.

- Process Module Qualifications
 - Process Module Tool Qualification
 - Process Module Execute Tests
 - Process Module Document Qualification
 - Process Module Compiler Qualification : Tool Qualification
 - Process Module TCA Qualification : Tool Qualification
 - Process Module Run Tests Manually : Apply QST without TAU
 - Process Module Apply QST with TAU : Tool Qualification
 - Process Module Run TAU : Execute Tests
 - Process Module Generate TQR : Document Qualification
 - Process Module Apply QST without TAU : Apply QST with TAU
 - Process Module Run Mixed Tests <TEST:TestType>
 - Process Module Run a Test using TAU <TEST:TestType>
 - Process Module Run a Test Manually <TEST:TestType> : Run a Test using TAU
 - Process Module Prepare Test
 - Parameters
- Artifacts
 - Artifact QKit
 - Artifact Language Suite : Test Suite
 - Artifact Tool Qualification Report
 - Artifact Compiler Qualification Report : Tool Qualification Report
 - Artifact Test Report
 - Artifact Human Tester : TAU
 - Artifact Excel Test List : Test Report
- Stake Holders
 - Stake Holder Qualifier
- Types

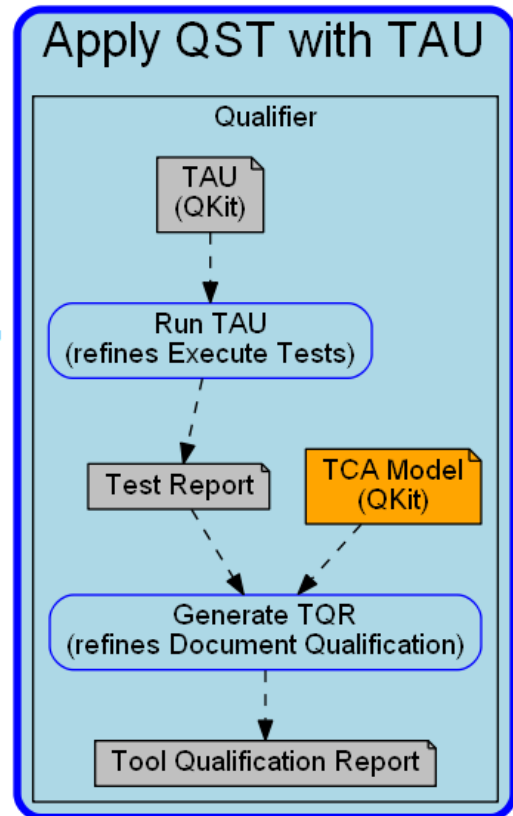


Figure 98: Refined Process Module Example

Since both sub-modules are replaced there is no difference (except with the interfaces) in this example. More interesting is the example Process Module "TCA Qualification", which refines the "Apply QST with TAU" and is re-using (using "SubProcessModuleReferences") a manual test execution module "Run Tests Manually" instead of the inherited process module "Run TAU", see Figure 99.

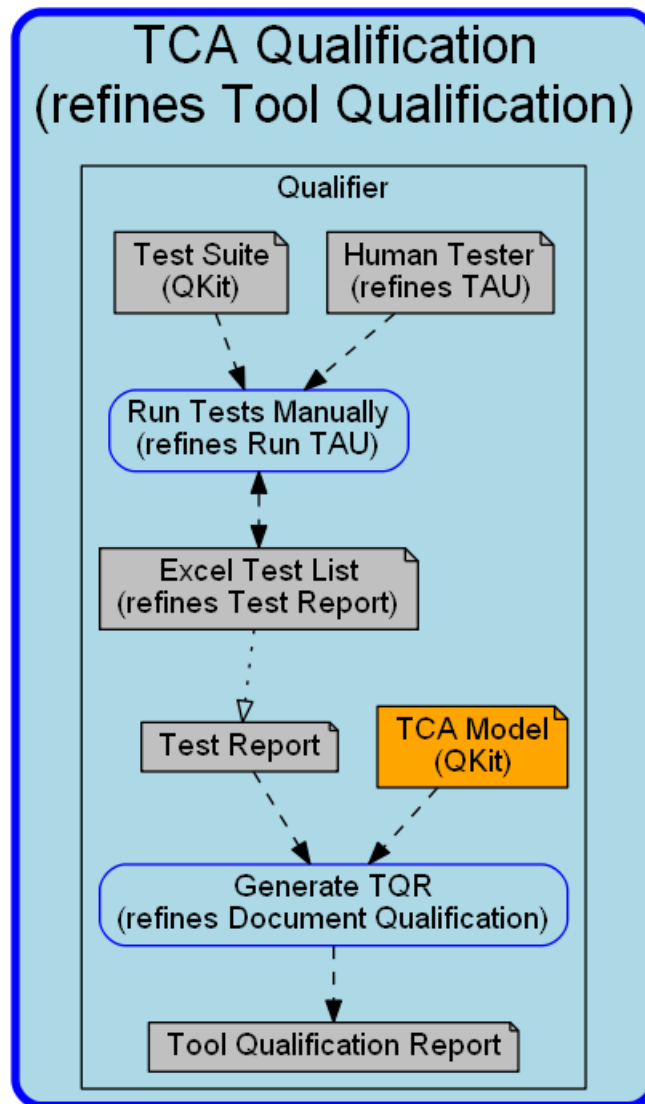


Figure 99: Complex Refined Process Module Example

The generated process report shows all sub-processes correctly, see Figure 100.

8.5 Validation

PMT supports three forms of validations that are described here:

- Syntactic / Automated Validation: Validates the model for selected consistency checks
- Graphical Validation: Shows the process graphically and allows the user to detect anomalies immediately

- Semantic Validation: This is a detailed review according to the so called meta-process of PMT that can be performed using Excel and VVT (as any other verification and validation planned using PMT)

8.5.1 Syntactic Validation

Syntactic validation applies automated consistency rules to the selected model element and their children in the tree (it does not consider references). It can be started using the Validate action on any element in the tree browser, see Figure 100.

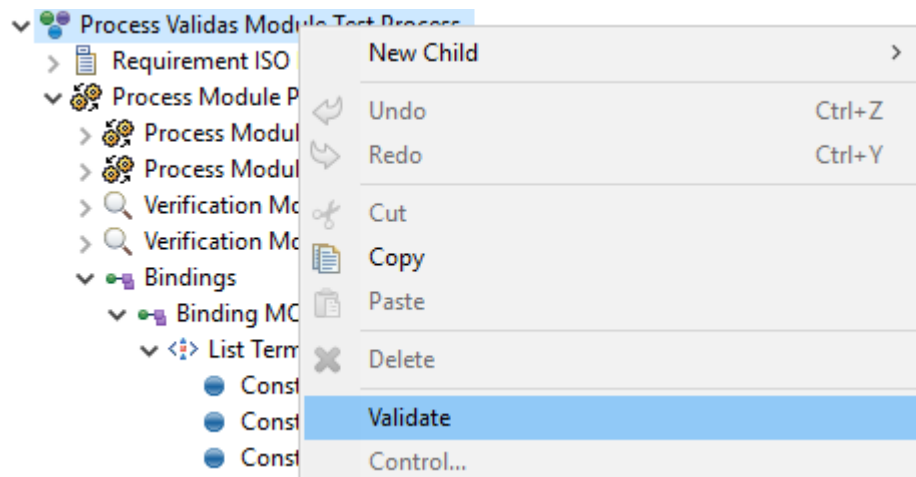


Figure 100: Starting the Validation

The results of the validation are shown in a dialog (see Figure 101) and in the Diagnostic View as described in Section 7.6.

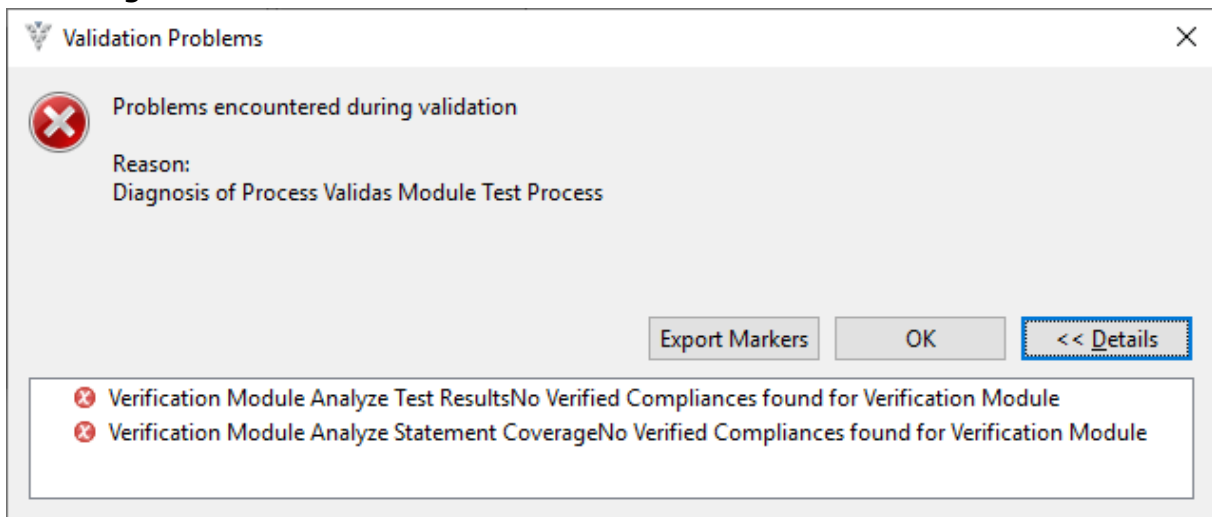


Figure 101: Validation Result Dialog

The following validation rules can be configured using the preferences mechanism from Section 8.8, as shown in Figure 102. Note that Validation configurations can be exported into preference files and reloaded such that the preferences can be harmonized within a given project where several users are working together.

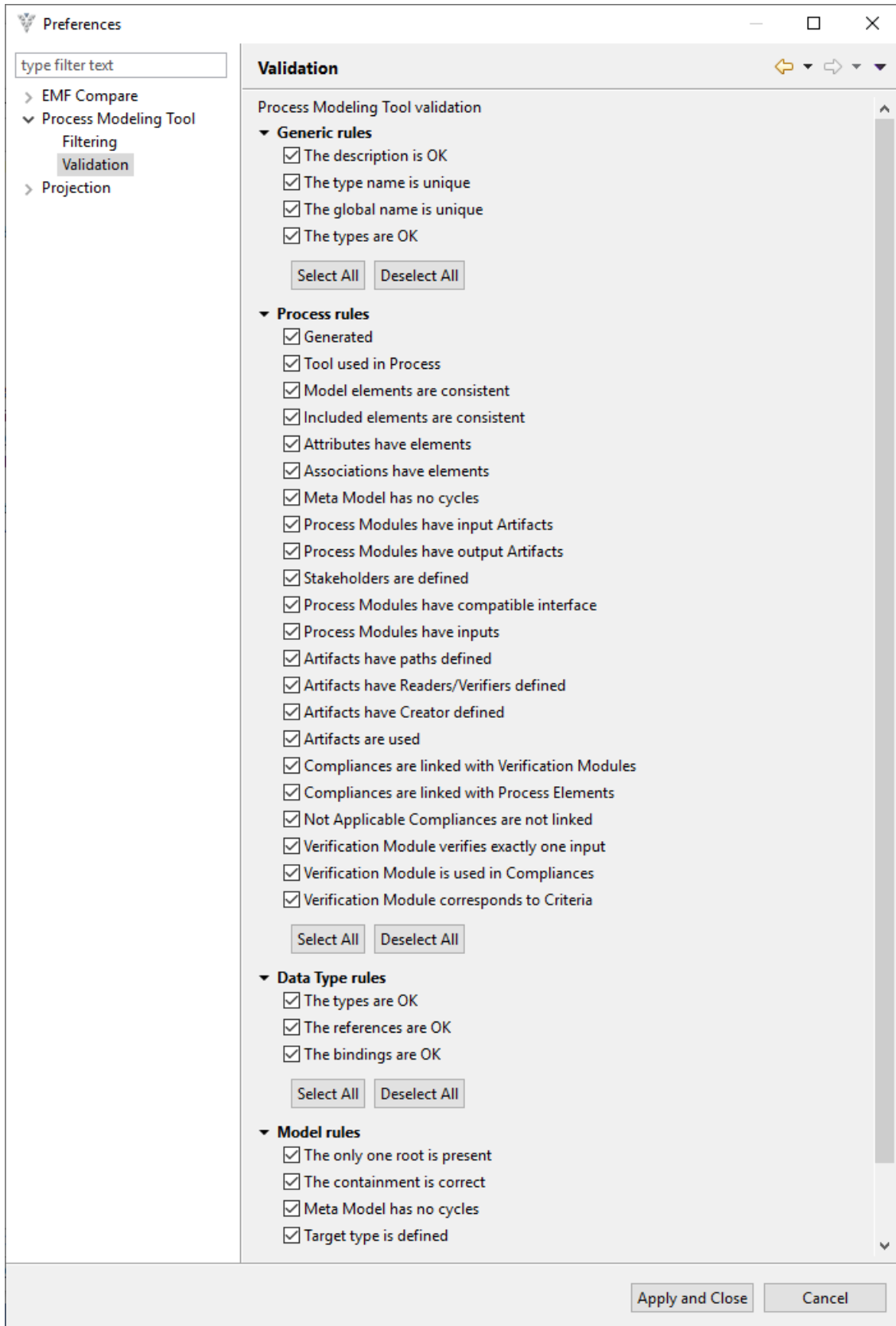


Figure 102: Configuration of Syntactic Validation Rules

8.5.2 Graphical Validation

„Only Nice Processes are good processes“, at least in the sense of clarity and understandability this is definitely true. This is the principle of the graphical validation. Graphical validation is done using the graphical Process View that shows the process graphically. Typical defects, like wrong sequencing or missing inputs/outputs can be seen immediately.

Note that for graphical validation graphviz is required (see Section 6) and the process modules have to have owners.

A graphically validated process is on the title page of this document.

8.5.3 Semantic Validation

The semantic validation is an intensive manual review of the process, guided by the meta-process and supported from PMT, Excel and VVT. Most details can be found in the MetaProcess.pmt description, which is included in the example folder of the PMT.

The core idea is that the meta-process is parameterized by the process modeling elements of PMT, i.e. it has a Parameter “VerificationModule” and a corresponding parameterized VerificationModule that iterates over all Verification modules that the user has modeled. The list of these parameter instances is exported from PMT. It is the first (and only) step of the semantic validation done in PMT, see Section 8.6.3.3.

8.6 Interfaces

PMT tool has several interfaces that are described in this section.

8.6.1 ProcessModule Ex- and Import

PMT allows to export ProcessModule elements into PMT files. By doing this PMT models can be copied, moved, merged etc. in a modular way. The export can be started using the popup-action on ProcessModule elements: “Export ProcessModule (.pmt)”, see Figure 103.

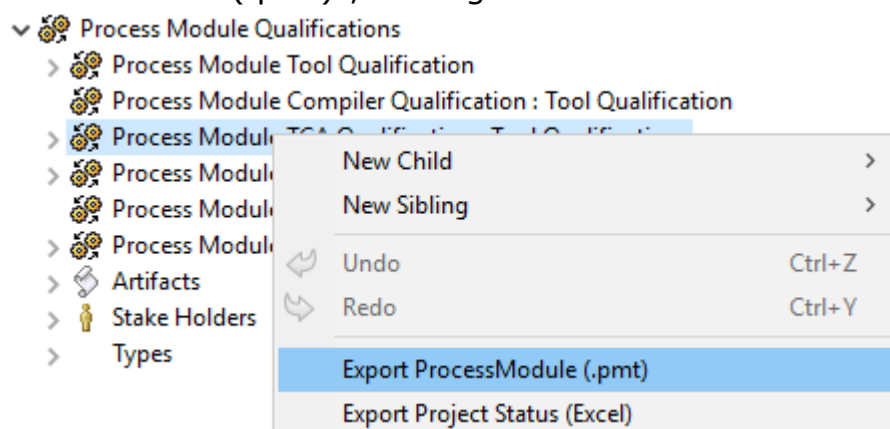


Figure 103: Starting PMT Export to File

The stored PMT file contains only the selected ProcessModule element(s) including all references and containers. It can be imported into any other PMF model (on the Process level) by starting the corresponding “Import ProcessModule (.pmt)” popup action on Process elements, see Figure 104.

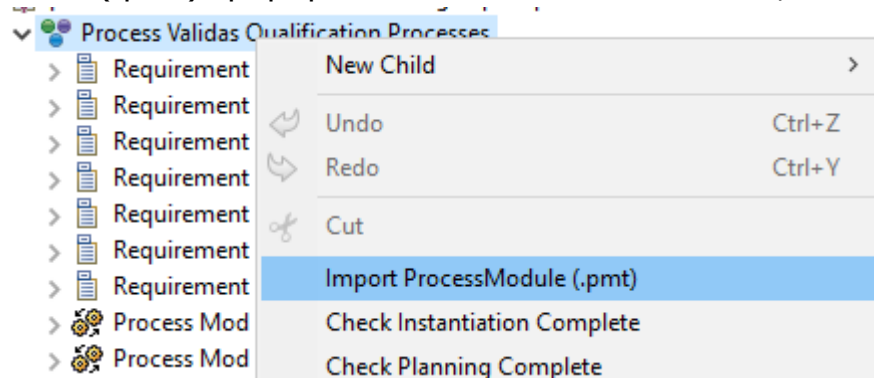


Figure 104: Starting PMT Import from File

After importing the model a result dialog shows the imported elements, see Figure 105.

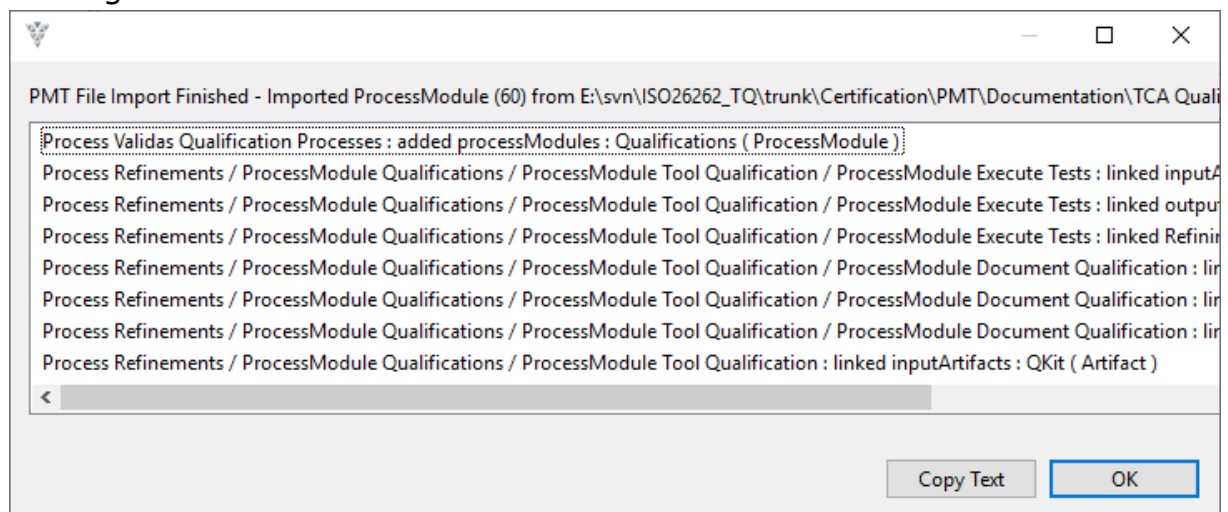


Figure 105: Import Log Messages

8.6.2 VVT

The Verification and Validation Tool (VVT) works on models with the extension .vvt. PMT can create an initial model, called “Schema” containing all selected verification modules with corresponding criteria. This schema then needs to be instantiated for every element in the current project. For example if you plan to test modules based on a verification process modeled using PMT, then the corresponding V&V Schema can be exported using the VVT interface. VVT can handle instantiation of the schema (to all test modules in the example) in a

similar way as PMT can instantiate processes to manage them in Section 8.2.7.

VVT Export can be triggered on Process and ProcessModules as shown in Figure 106

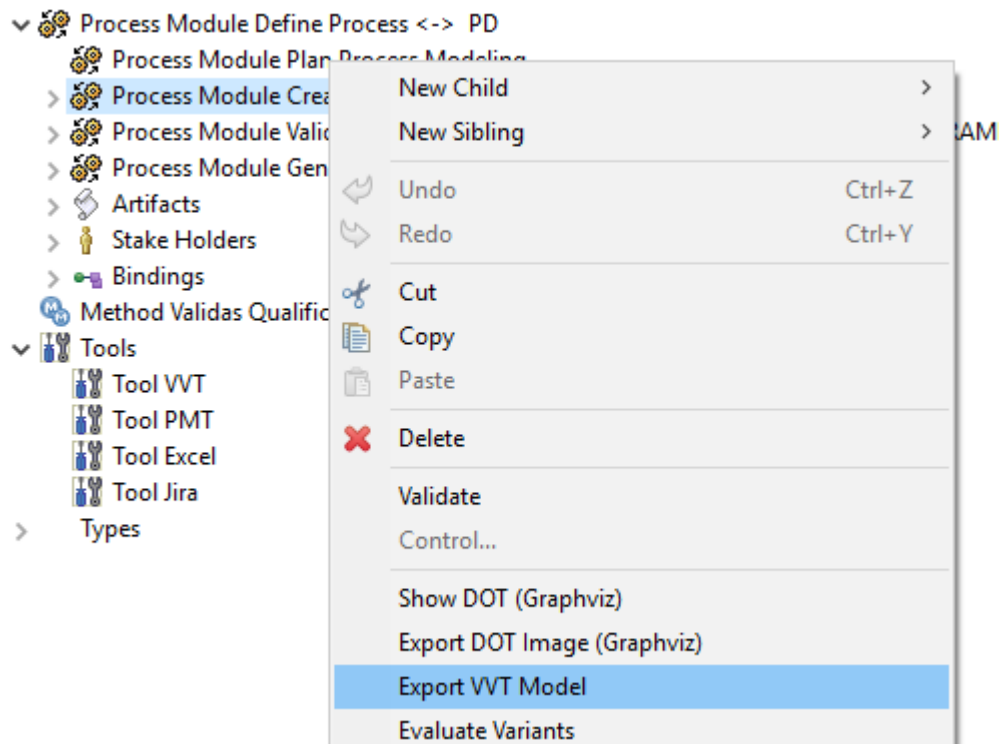


Figure 106: Start VVT Export

After selecting a destination (file) for the VVT model, the model is converted. After the export is finished, PMT shows a log file with the created checks, as shown in Figure 107.

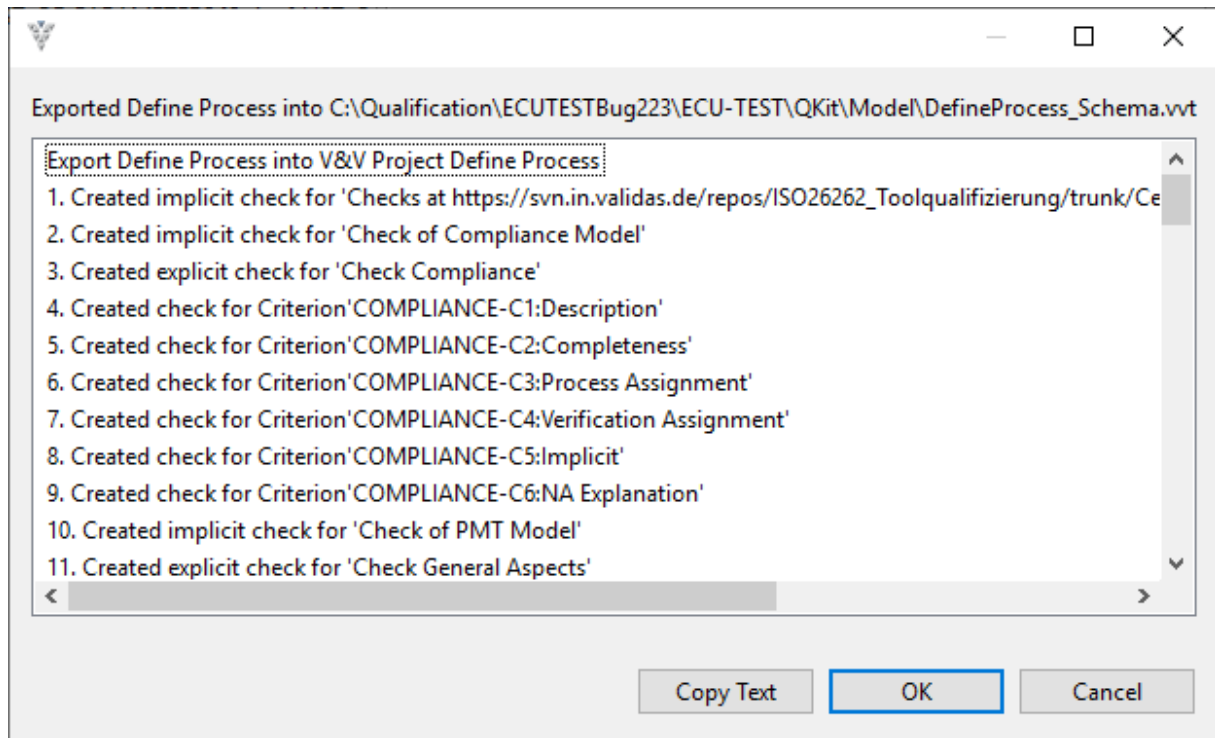


Figure 107: Log of VVT Export

8.6.3 Excel

PMT uses Excel for interfaces for the following purposes:

- Parameters for Instantiations
- Process Status
- Process Description (Export Only)

8.6.3.1 Parameters Interface

Parameters can be exported using the popup action on Process and ProcessModule elements (see Figure 108).

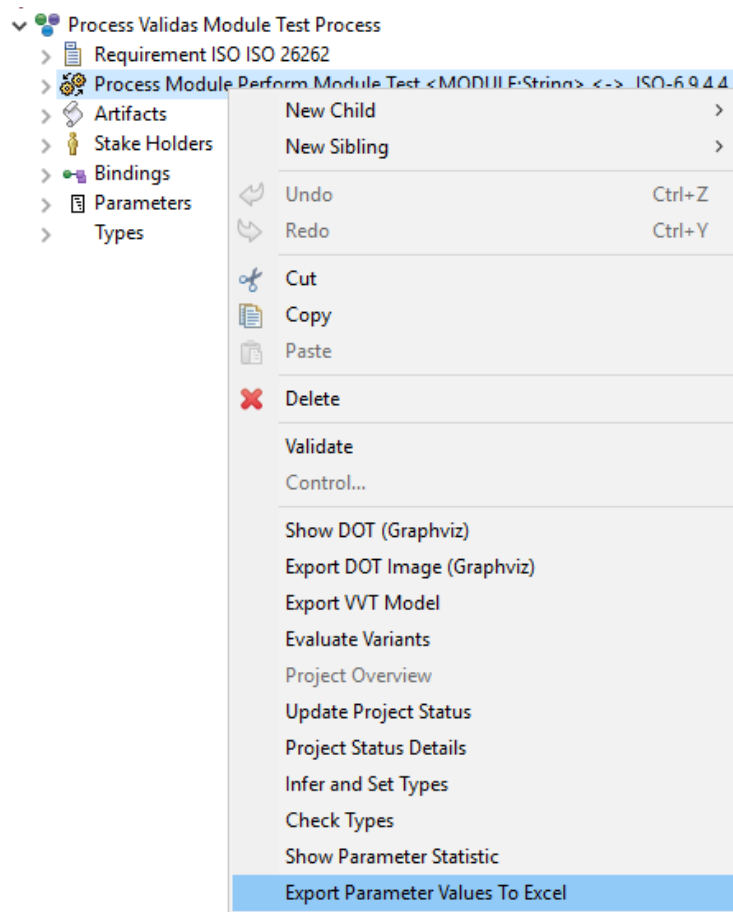


Figure 108: Start Parameter Export

The exported parameter table in Excel contains for each exported parameter a tab into which the values can be entered as shown in Figure 109.

	A	B	C
		Parameter Type	Container
1	Parameter Value		
2	main.c		
3	lib.c		
4	test.c		
5			
6			
7			

Figure 109: Parameter Values in Excel

After filling in some data (as shown in

Figure 109), the parameter values can be imported into PMT using the corresponding import actions, see Figure 110. Figure 111 shows a log from importing describing the performed changes. Note that the import also check the types of the values match with the exported types from the parameters.

The results of the import are new / updated Parameter Bindings (see Figure 112) that can be used to instantiate the process as described in Section 8.2.7.

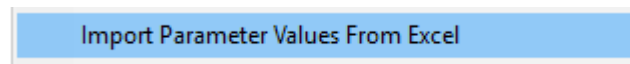


Figure 110: Starting Parameter Value Import From Excel

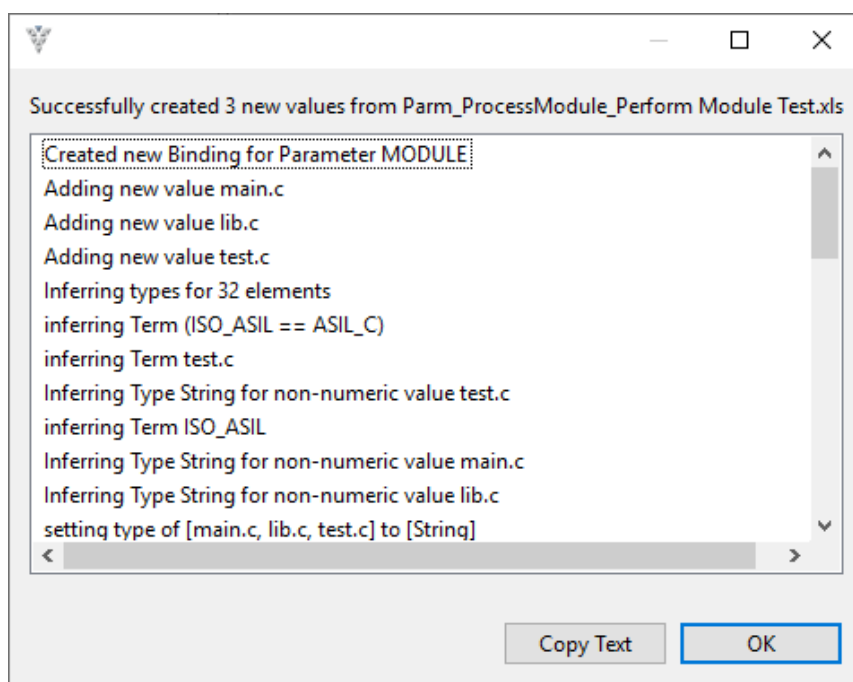


Figure 111: Log from Parameter Value Import

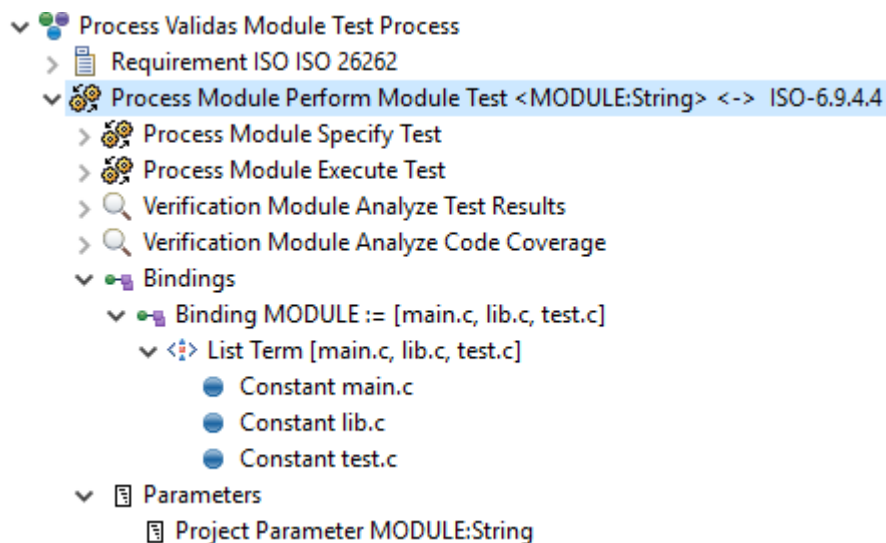


Figure 112: Resulting Parameter Binding from Parameter Value Import

8.6.3.2 Process Status

The process status can be used to trac/manage the project status, see Section 8.2.11. PMT support this by providing an Excel format Export / Import of the status of the artifacts and ProcessModules/Verification Modules.

The ex- and import can be started from ProcessModule (and VerificationModule) elements as shown in Figure 113.

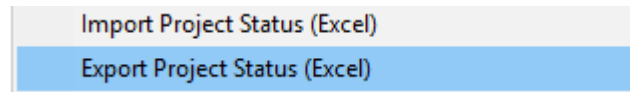


Figure 113: Project Status Excel Interface Actions

The resulting Excel table (that can be updated and re-imported) is shown in Figure 114. It has rows for all artifacts and ProcessModuels with the following information:

- Type: The type of the element (Artifact or ProcessModule / VerificationModule).
- Name: The name of the element.
- State: The status of the elements, see Section 9.2.1 for all available states.
- The ID of the element.
- The qualified name of the element (do not change this, since this is used to locate the elements in the model).
- Description: A description of the element.
- Effort: a number for the estimated effort (can be hours, days or \$).
- Progress: a number normal less or equal the estimated effort.
- Planned Start: A date when the work should start.
- Planned End: A date when the work should end.

Please note that PMT will not remove elements if you delete rows in your excel sheet. This has to be done automatically.

	A	B	C	D	E	F	G	H	I	J
		Name	State	ID	Qualified Name	Description	Effort	Progress	Planned Start	Planned End
1	Type									
2	Artifact	SUT	DEFINED	SUT		Subject Under Test, t				
3	Artifact	Safety Case	DEFINED	Safety Case		Containing all relevan				
4	Artifact	Code Coverage Report	DEFINED	Safety Case.Code Coverage Report		The report with the se				
5	Artifact	Summary	DEFINED	Safety Case.Summary		ModuleTest/Artifacts/				
6	Artifact	Test Case	DEFINED	Safety Case.Test Case		A module test case (
7	Artifact	Test Report	DEFINED	Safety Case.Test Report		The test report create				
8	Artifact	Specification	DEFINED	Specification		Functional Specificat				
9	Artifact	Test Specification	DEFINED	Test Specification		Model based test spe				
10	ProcessModule	Perform Module Test	DEFINED	Perform Module Test		All activities for perfor	0	0		
11	VerificationModule	Analyze Code Coverage	DEFINED	Perform Module Test.Analyze Code Coverage		The code coverage is	0	0		
12	VerificationModule	Analyze MCDC	DEFINED	Perform Module Test.Analyze Code Coverage.Analyze MCDC		Analyze if MCDC is	0	0		
13	VerificationModule	Analyze Test Results	DEFINED	Perform Module Test.Analyze Test Results		Checks the test resu	0	0		
14	ProcessModule	Execute Test	DEFINED	Perform Module Test.Execute Test		Run the test cases a	0	0		
15	ProcessModule	Measure MCDC	DEFINED	Perform Module Test.Execute Test.Measure MCDC		Perform the tests anc	0	0		
16	ProcessModule	Run Test	DEFINED	Perform Module Test.Execute Test.Run Test		Executes test case a	0	0		
17	ProcessModule	Specify Test	DEFINED	Perform Module Test.Specify Test		Create a model base	0	0		
18	ProcessModule	Generate Tests	DEFINED	Perform Module Test.Specify Test.Generate Tests		Generate tests cases	0	0		
19	ProcessModule	Model Test Behaviour	DEFINED	Perform Module Test.Specify Test.Model Test Behaviour		Create a model for th	0	0		
20	VerificationModule	Validate Tests	DEFINED	Perform Module Test.Specify Test.Validate Tests		Tests can be validate	0	0		
21										
22										

Figure 114: Project Status Format in Excel

The import shows a log file including the detected changes and warnings as depicted in Figure 115.

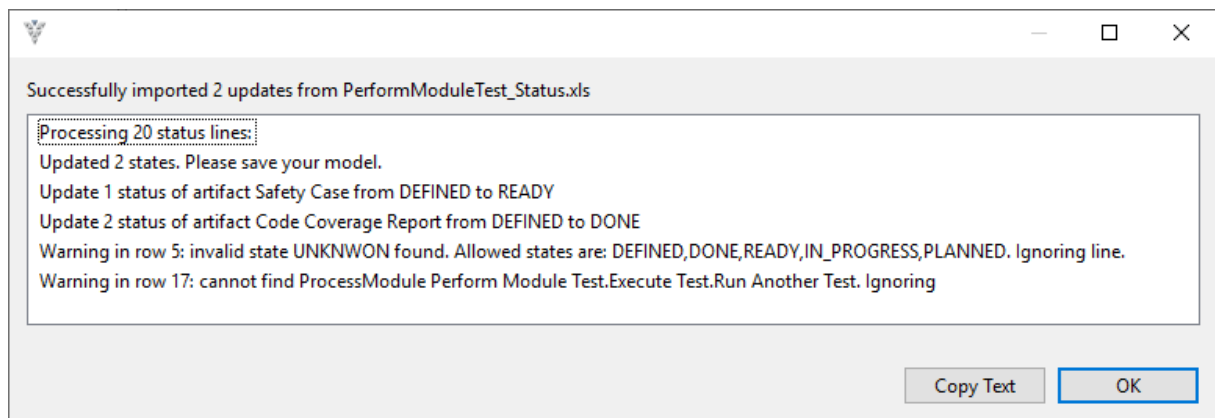


Figure 115: Log from Importing Project Status from Excel

8.6.3.3 Process Description Export

In order to semantically validate a process the so called MetaProcess has to be applied, see Section 8.5.3.

The export of the process description is started on ProcessModule elements as shown in Figure 116.

Export Process Parameters (Excel)

Figure 116: Starting Export Process Parameter into Excel (ProcessModule)

The result is in the same format as the parameter export described in Section 8.6.3.1, see Figure 117 for an example.

Note that not only the selected Process will be exported, but also the linked elements like requirements, compliance argumentation,...

	A	B	C	D	E	F	G
		Parameter Type	Container				
1	Parameter Value						
2	Analyze Code Coverage	String	Perform Module Test				
3	Analyze MCDC	String	Analyze Code Coverage (in Perform Module Test)				
4	Analyze Test Results	String	Perform Module Test				
5	Execute Test	String	Perform Module Test				
6	Generate Tests	String	Specify Test (in Perform Module Test)				
7	Measure MCDC	String	Execute Test (in Perform Module Test)				
8	Model Test Behaviour	String	Specify Test (in Perform Module Test)				
9	Perform Module Test	String	Validas Module Test Process				
10	Run Test	String	Execute Test (in Perform Module Test)				
11	Specify Test	String	Perform Module Test				
12	Validate Tests	String	Specify Test (in Perform Module Test)				
13							

COMPLIANCE PARAMETER **PROCESS** PROCESS_ARTIFACT REQUIREMENT ROLE TOOL ...

Figure 117: Process Description in Parameter Value Format

The exported excel table contains the following tabs, each filled (or empty) with the used modeling elements:

- COMPLIANCE: the compliance elements relevant for the selected process
- PARAMETER: the used parameters in the model (including referenced ones)
- PROCESS: The selected process modules (and the referenced children), but no VerificationModules
- PROCESS_ARTIFACT: The artifacts in the process
- REQUIREMENT: The requirements satisfied by the process
- ROLE: the used StakeHolders
- TOOL: the used tools
- VARIANT: the used variant terms
- VERIFICATION: the included verification modules

More detailed description of the parameters can be found in the MetaProcess.pmt in the examples folder of PMT.

8.6.3.4 Development Interface Agreement

In order to cooperate in safety related process it is important to agree on a work split. This can be done using excel tables which contain all artifacts of the process. This table can be generated using the excel export on Project Modules using the

Figure 118: Starting Export Development Interface Agreement (ProcessModule)

The result is an excel table with the following tables:

- Overview: describes the document status, version, author and history
- Artifacts: The list of all artifacts with the informations from the model
- Roles: the roles in the process
- Parameters: the used parameters
- Steps: the used process & verification modules

The table cannot be imported again into the model.

8.6.3.5 Offer

This action (see Figure 119) exports a calculation for all artifacts that have to be delivered from the selected roles (first dialog, see Figure 120)

Figure 119: Starting Export of Offer (ProcessModule)

The offer selection dialog shows all roles which are responsible for artifacts in the selected process and allows to select several roles.

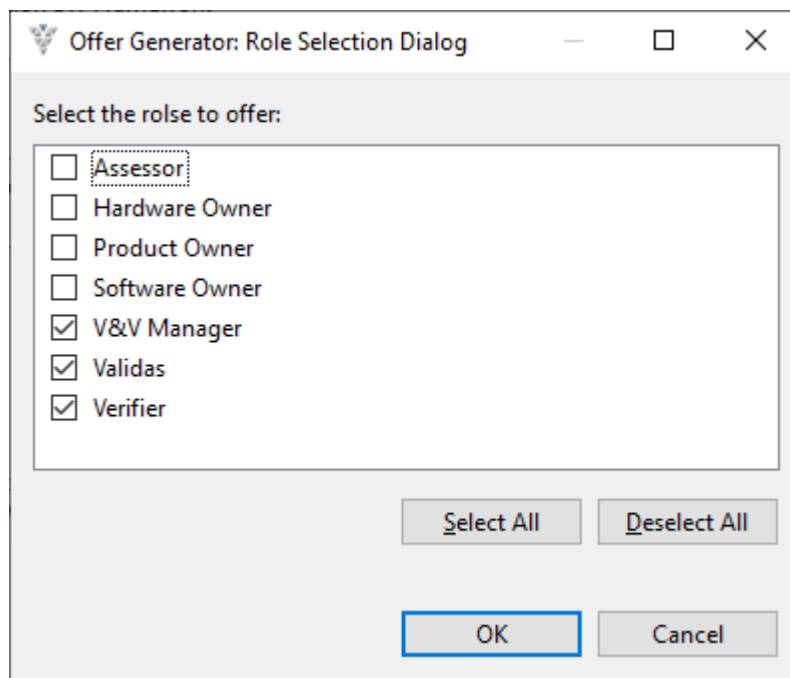


Figure 120: Offer Selection Dialog

The result is an excel table with a calculation for prices of all artifacts (that have ProjectRelevant set to true and that are in the responsibility of the selected roles.). The prices are computed based on the daily rate and the effort (multiplied by "NumberOfInstance" in case the artifact is required

for several parameter values). If no efforts are specified, the default value of 2 is used for all artifacts that have no relation to a compliance element for a requirement and 5 if the document is required for compliance argumentation.

Figure 121: Daily Rate Dialog

The daily rate has to be entered (as positive number) within the next dialog, see Figure 121

The generated calculation table cannot be imported again into the model.

F	M	N	P
Deliverable	PD	Sum WP	Price
		86	129000
Artifact Variables [6-C]	5		7500
CheckList	2		3000
CheckResult	2		3000
DepFailR [6-7]	5		7500
Environment Parameters [6-C]	5		7500
Features	2		3000
HSI Spec (refined) [6-6]	5		7500
Mitigations	2		3000
Parameters	2		3000
Potential Errors	2		3000
SW Arch DS [6-7, 6-C]	5		7500
SW Safety RS [6-6]	5		7500
SWA Ana Report [6-7]	5		7500

Figure 122: Calculation Table (partly)

8.6.4 Ecore Importer

PMT allows to create and use a meta model for the definition of model-based processes, see Section 8.2.2 and 9.12.

To reduce the work for modeling tools based on Eclipse Modeling Framework using an .ecore file, the meta model of a tool can be imported automatically from that file into the model.

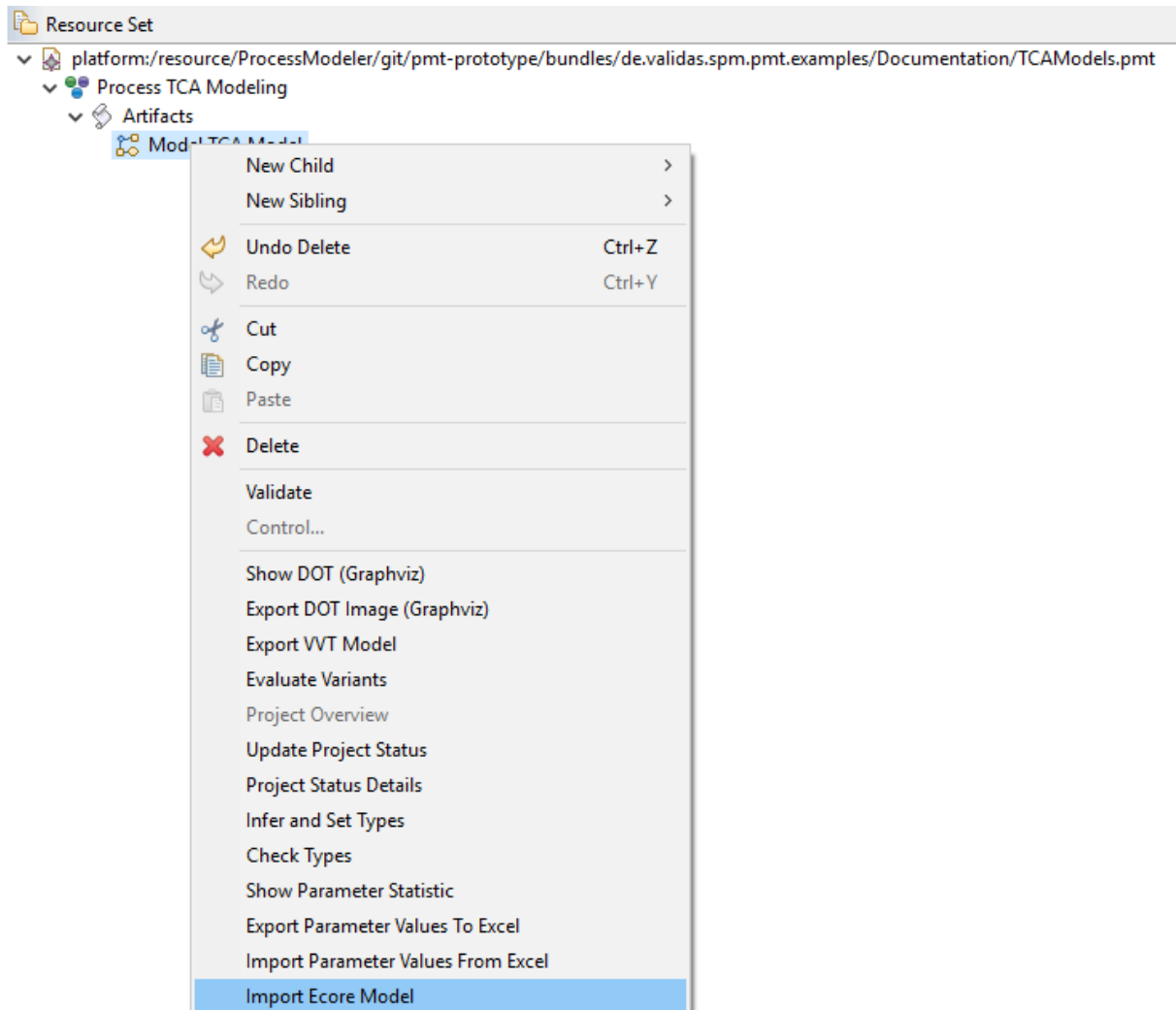


Figure 123: Starting Import Ecore Model

After selecting an .ecore file the meta model is imported and a log file is displayed as shown in Figure 124.

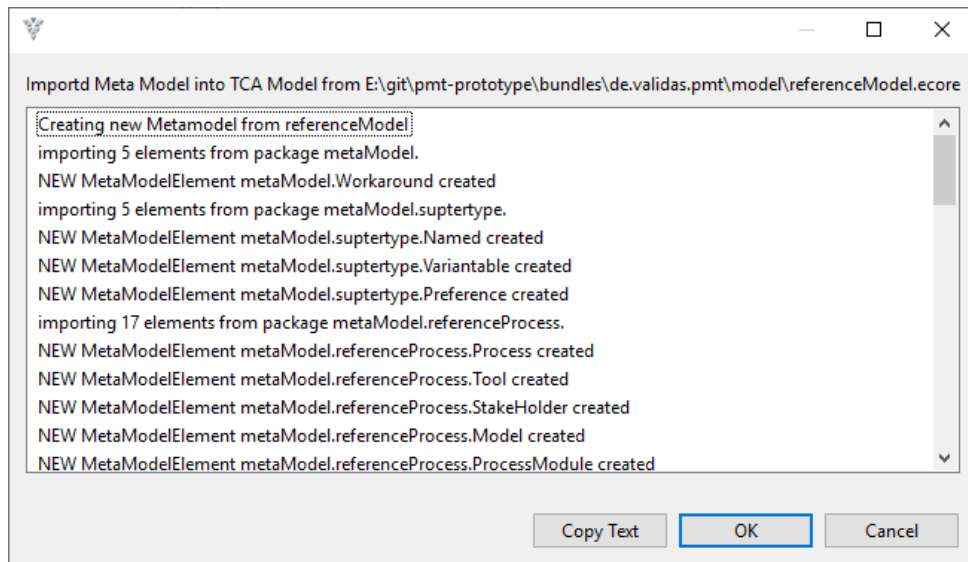


Figure 124: Import Log from Ecore Importer

8.7 Report Generators

PMT supports the generation of the following reports:

- Process Report: contains a detailed description of the process.
- Compliance Report: contains the compliance argumentation.

Both report generators use templates that are included in the PM tool but can be adapted if desired.

Note: both reports contain variables that need to be updated manually within Word by selecting the whole document (STRG+A) and updating the variables (F9).

8.7.1 Process Report

The process report can be generated from any ProcessModule and Process by starting the action "Word Generators -> Process Report" as shown in Figure 125.



Figure 125: Starting Process Report Generation

The generated report has the structure and title pages as shown in Figure 126.

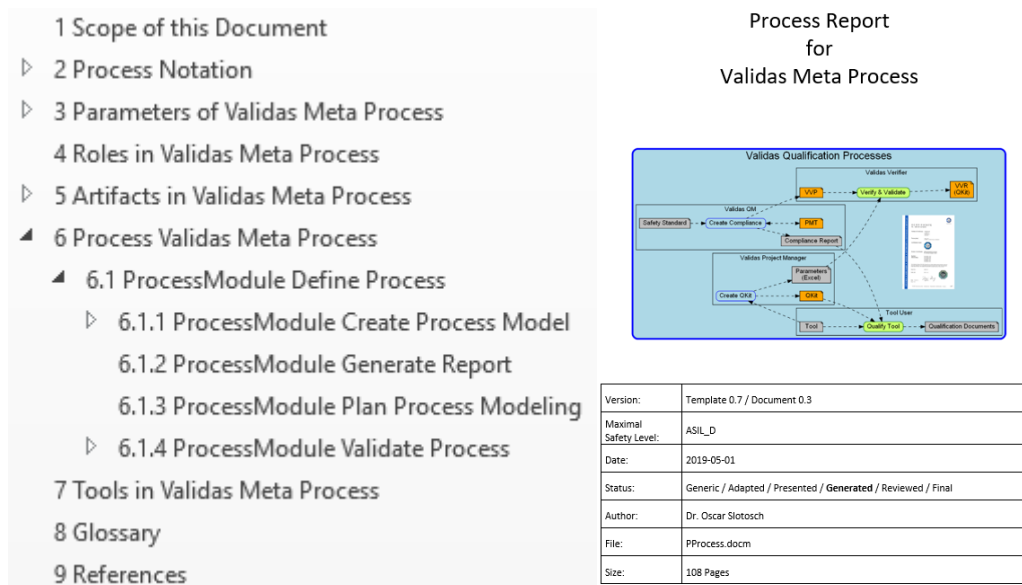


Figure 126: Structure and Title of Process Report

8.7.2 Compliance Report

The compliance report can be generated from any ProcessModule and Process by starting the action “Word Generators -> Compliance Report” as shown in Figure 127.

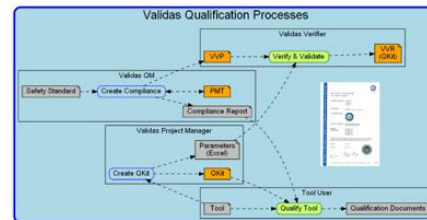


Figure 127: Starting Compliance Report Generation

The generated report has the structure and title pages as shown in Figure 128.

1	Scope of this Document
2	Compliance Method
3	Parameters of Define Process
4	Main Requirements (Claims) for Define Process
5	Requirements for Define Process
5.1	Requirement: Process Description Requirements [PD]
5.1.1	Requirement: Artifacts and Documents [PD-Docs]
5.1.2	Requirement: Open Issues [PD-Issues]
5.1.3	Requirement: Parameters and Instances [PD-Param]
5.1.4	Requirement: Process and Verification Steps [PD-Proc]
5.1.5	Requirement: Requirements and Compliances [PD-Req]
5.1.6	Requirement: Roles and Responsibilities [PD-Roles]
5.1.7	Requirement: Tool Support [PD-Tools]
5.1.8	Requirement: Variants and Tailoring [PD-Variant]
6	V&V Checks for Define Process
7	Compliance for Define Process
7.1	Compliance with Process Description Requirements [PD]
7.1.1	Compliance with Artifacts and Documents [PD-Docs]
7.1.2	Compliance with Open Issues [PD-Issues]
7.1.3	Compliance with Parameters and Instances [PD-Param]
7.1.4	Compliance with Process and Verification Steps [PD-Proc]
7.1.5	Compliance with Requirements and Compliances [PD-Req]
7.1.6	Compliance with Roles and Responsibilities [PD-Roles]
7.1.7	Compliance with Tool Support [PD-Tools]
7.1.8	Compliance with Variants and Tailoring [PD-Variant]
8	Glossary
9	References

Compliance Report for Define Process



Version:	Template 0.7 / Document 0.3
Maximal Safety Level:	ASIL_D
Date:	2019-05-01
Status:	Generic / Generated / Reviewed / Final
Author:	Dr. Oscar Slotosch
File:	CPM.docm
Size:	55 Pages

Figure 128: Structure and Title of Compliance Report

8.8 Preferences

PMT has the following preferences that can be adjusted (see Figure 129):

- General Preferences: Define the appearance of PMT property editors:
- Filtering Preferences: Configure the filtering mechanism, see Section 8.9.
- Validation Preferences: Select the syntactic validation rules see Section 8.5.1.
- Projection Preferences: Configure the model projection, see Section 7.5.

Qualified names reflect the tree hierarchy by concatenating all name segments using the configured separation character.

Qualified names can be very valuable in bigger models to find the right element in long element lists of “selection dialogs”, where you want to select an element, for example a process that creates an artifact.

Qualified names can also be used in the property dialogs, if configured.

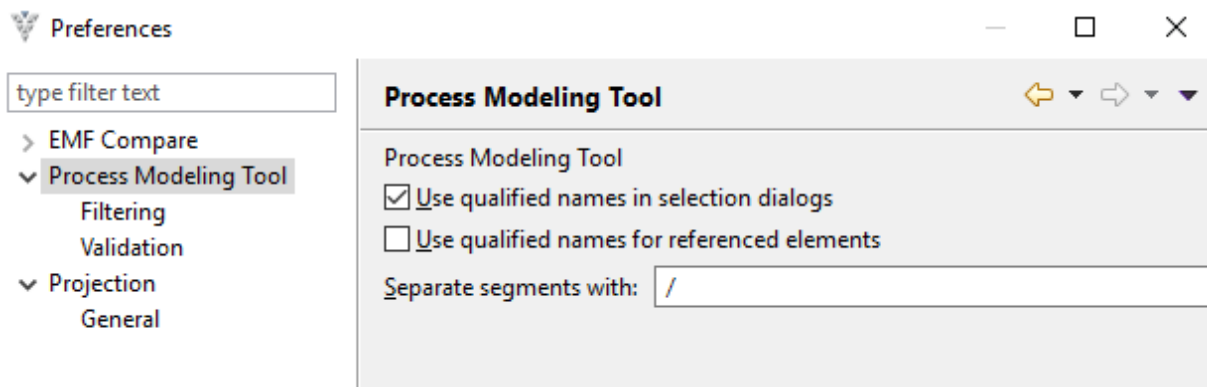


Figure 129: PMT Preference Window

8.9 Filter Scoping

In order to reduce the number of displayed elements in the dialogs scopes can be used and activated.

A scope is a process module and all contained elements. Scopes are stored in the Process element (root container) in order to make them part of the model.

The filter scopes have to be activated within the preference dialog by selecting "Active Process Modules" in the "Active Scope" dialog as shown in Figure 130

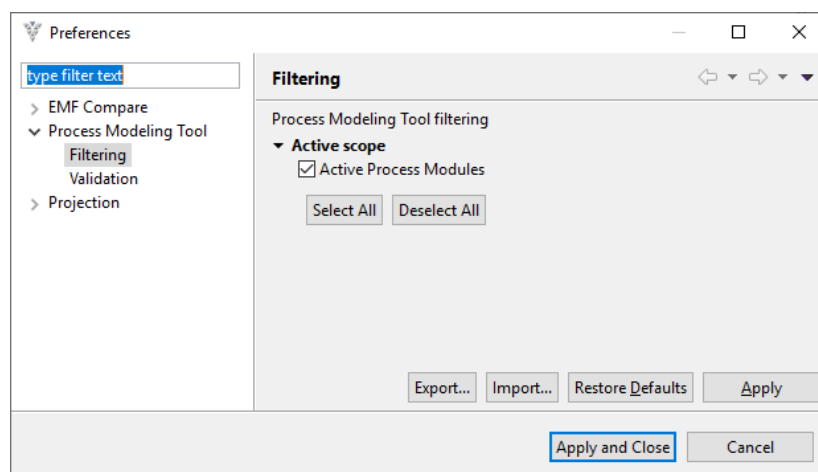


Figure 130: Activation of Filter Scoping

Once activated the Scopes can be selected as shown in Figure 131.

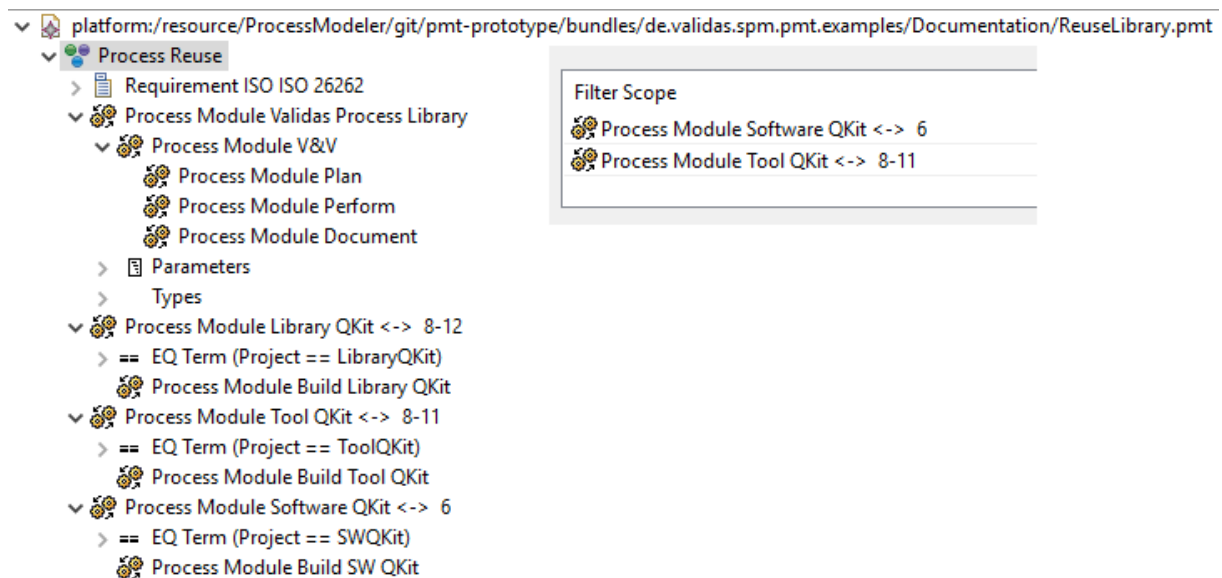


Figure 131: Definition of Filter Scopes

Figure 132 shows the result in the selection dialog. Only the filtered elements are shown (and the top-level process modules).

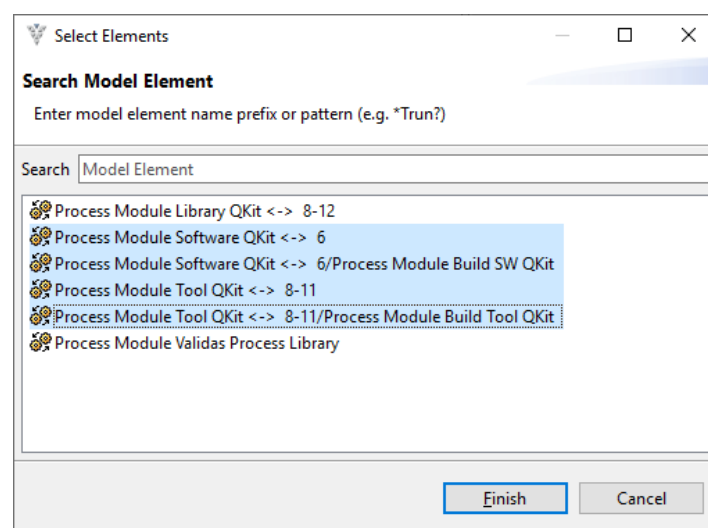


Figure 132: Application of Filter Scopes

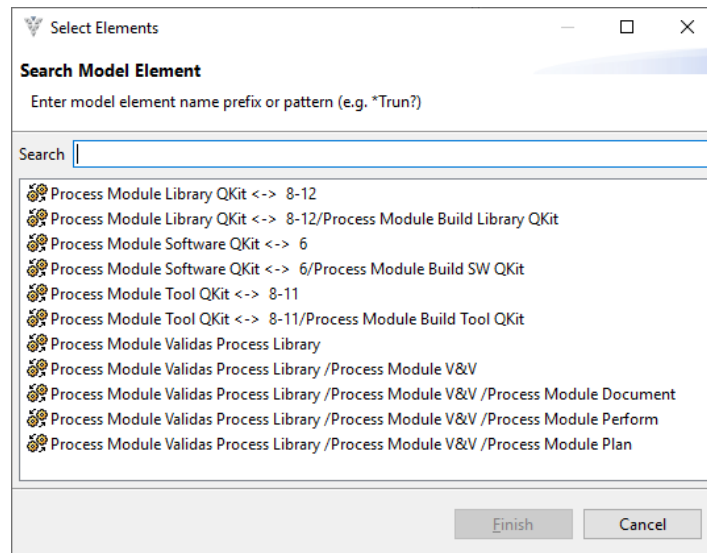


Figure 133: Application without Filter Scopes

9 Meta Model of PMT

This chapter describes the complete process model and the elements together with modeling rules and the visualization in the browser.

A coarse overview of the model ("powerpoint level") is depicted in Figure 134. It shows the main structures but omits the details that will be described in the remainder of this section.

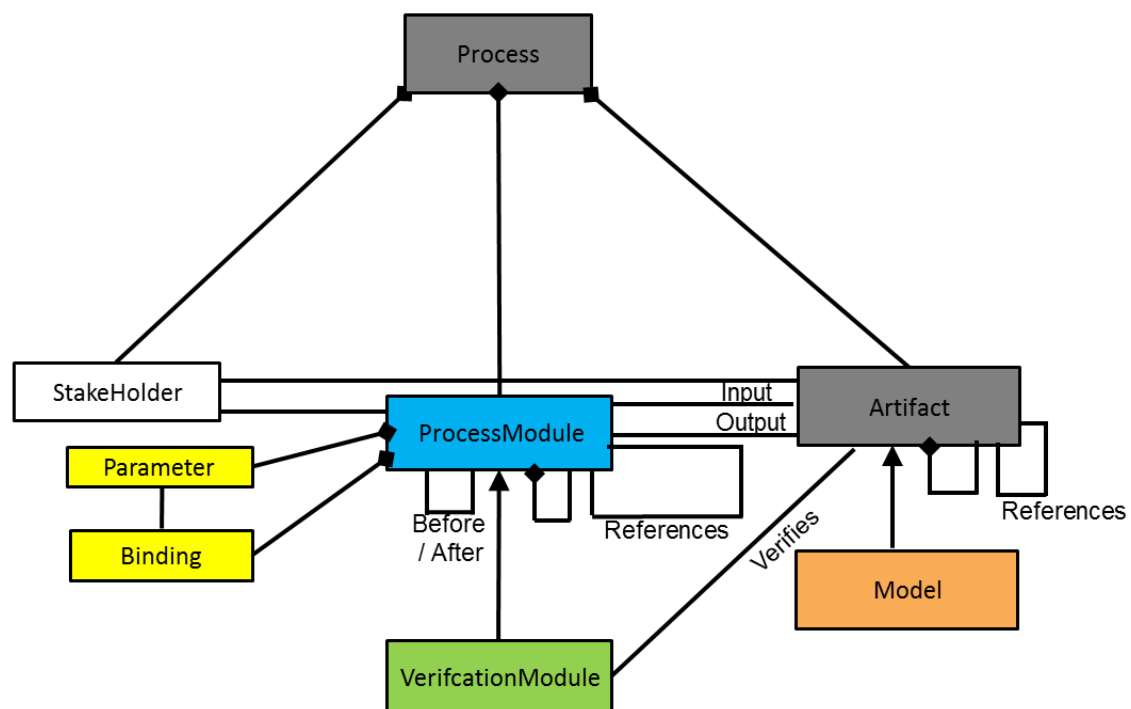


Figure 134: Metamodel - Powerpoint Level

The PMT is an Eclipse-based RCP application. The core of the application is the meta-model. Section 9.1 describes the basic principles. The model has common interface, see Section 9.3 and follows some scoping rules (see Section 9.4).

The model is described in semantic groups (class diagrams) that focus on semantic aspects, e.g. tailoring, tools,.. The class diagrams focus on those aspects. However the descriptions of the classes contain all aspects such that it can be used as reference.

9.1 Syntax of Meta Model

The meta model describes the structure of the model. It defines the syntax, i.e. the form of the model. The following chapters describe the interpretation and semantics of the model.

The meta model consists of the following elements:

- Classes with names
- Attributes (of classes) with names and types
- Associations to other classes with names and cardinalities

The meta model describes the possible models as follows:

- Every element of the model has to be of a given type which is represented as class in the meta model
- Every element in the model has properties that can be changed by the user. Properties are either attributes, e.g. a name or relations ("associations") or "compositions" to other elements in the model.
- Attributes have types, e.g. String, Integer, Boolean to constraint the possible elements that can be modeled.
- Associations are links to other elements in the model of a given type. Associations have cardinalities to specify the number of associated elements. Used cardinalities are in PMT
 - 0..1: zero or one occurrence
 - 0..*: zero or arbitrary
- Compositions are like associations, except that the related elements are "contained" in the element, i.e. are deleted when the element is deleted.
- Containers: can contain the elements, i.e. if an element has to be created this can only be done within an appropriate container.

The meta model is implemented within an ecore file in PMT and described using class diagrams. Classes are depicted as boxes with their names on top and the attributes and their types after a colon below the line. Relations are depicted as arrows. Their names and cardinalities are depicted at the end of the outgoing arrows. Some arrows are bi-directed, denoting that the relation between the objects is "inverse", i.e. every object A has relation to B and B has inverse relation to A. The smart thing is that it suffices to specify one of both associations and the opposite is set automatically from the tool. Compositions are marked with black diamonds on the container.

Figure 135 shows an example: It shows the classes "Process", "Tool" and "ProcessModule", their attributes and relations. There is a composition

from Process to Tool called “tools”, denoting that the process contains Tool elements and that they are called tools. The element “Tool” has two attributes of type String: PreliminaryClassification and ClassificationExplanation, those can be used to describe a preliminary classification, e.g. “critical” or “uncritical” and a textual explanation. Furthermore the example specifies a bi-directional association between Tools and ProcessModules denoting that tools can be used in process modules and process modules can be supported by tools. All cardinalities are 0..*.

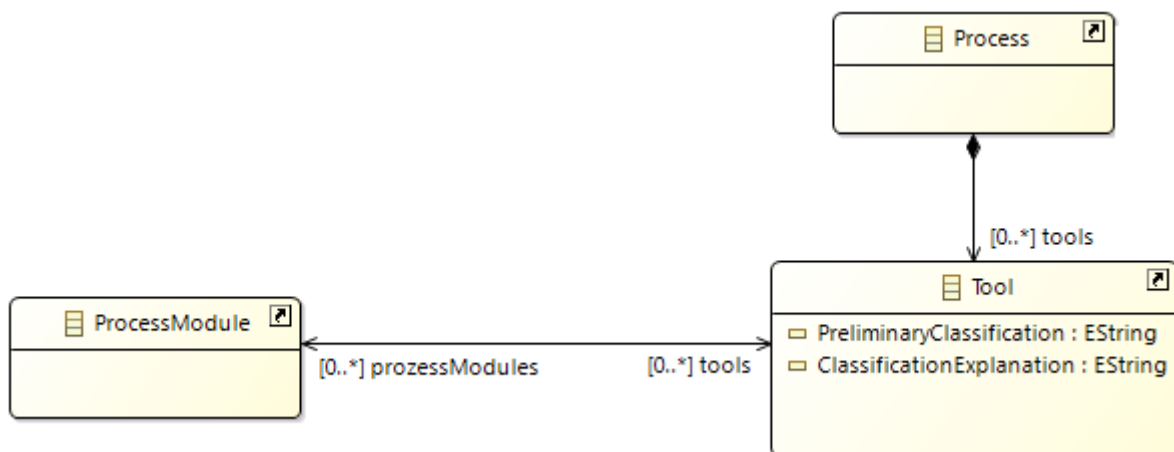


Figure 135: Meta Model: Class Diagram Example

The described meta model results into the following PMT models:

The compositions are structure in the tree browser. In Figure 136 it can be seen that the selected element “Test Tool” is contained in the process “Validas Module Test Process” (via the group Tools) and that it has (despite the general properties like Name, Description,...) the two specified attributes: “Preliminary Classification” and “Classification Explanation”.

Furthermore the “Test Tool” element has an association to the Process Module “Execute Test”.

Note that the class diagram might not show all properties to keep it simple (however the specifications in this section are complete), therefore there are also other properties of the tool that can be edited. For example there is also a “Tool Owner” that can be set to at most one StakeHolder (Cardinality is 0..1).

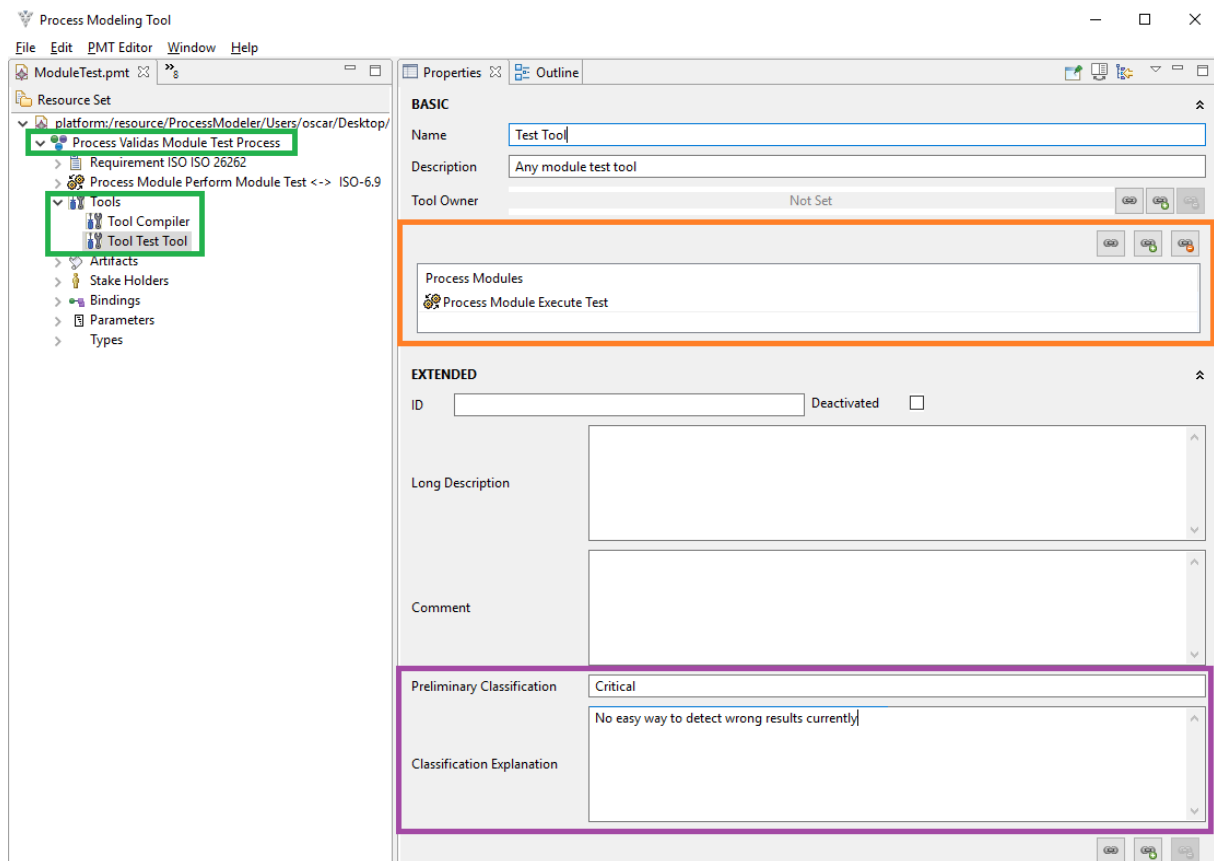


Figure 136: Model: PMT Structure Example

The meta model specifies the allowed models and their structures. Further information about these concepts can be found in the documentation of the used Eclipse Modeling Framework (EMF).

9.2 Enumerations

Some attributes have a finite set of values and are defined using enumerations. Those are specified in this section.

9.2.1 ProcessStatus

The process status in PMT consists of the following values:

- **DEFINED:** the task/artifact is defined, but not ready to start working.
- **PLANNED:** The task/artifact is planned, i.e. has resources and dates assigned.
- **READY:** the task/artifact is ready to start, i.e. its inputs are in the state DONE (Theoretically it would also be possible to work on inputs of status "IN_PROGRESS", but this is not useful for a formal calculus).
- **IN_PROGRESS:** the task/artifact has started but not ended.
- **DONE:** the task/artifact is done.

Note that the project status is defined for ProcessModules and Artifacts.

9.2.2 SafetyLevel

The SafetyLevel specifies the maximal safety level of the process. It can have the following values:

- **ASIL_A**
- **ASIL_B**
- **ASIL_C**
- **ASIL_D**
- **SIL_1**
- **SIL_2**
- **SIL_3**
- **SIL_4**
- **TQL_5**
- **TQL_4**
- **TQL_3**
- **TQL_2**
- **TQL_1**

9.2.3 Cardinality

The Cardinality is used to specify the number ("cardinality") of the elements that can be related to the element by these associations. Main cases in EMF are "one" or "many" associations. Cardinality can have the following values:

- **CARDINALITY_0_to_1**: Only one element can be present or not.
- **CARDINALITY_1**: Exact one element has to be present. Many tools do not enforce this condition, so it is rarely used.
- **CARDINALITY_0_to_N**: Many elements can be present or not.
- **CARDINALITY_0_to_N**: Many elements can be present but at least one has to be present. Many tools do not enforce this condition, so it is rarely used.

Also other cardinalities can be specified, e.g. "2" in UML, but for simplicity we decided to use a (finite) enumeration for cardinalities.

9.3 General Interfaces

In PMT there are two main interfaces that harmonize modeling, Figure 137: "Named" for all elements with names and "Variantable" for all

elements that can have tailoring using variant terms and graphical layouts⁶.

Figure 137 shows the interfaces, together with two example elements (Process and ProcessModule) and their relations. Note that “Variantable” is a specialization of “Named”, i.e. every element that can have variants/layouts is always “Named” with Name, Description,..

The “Process” class implements the Interface “Named”, hence it inherits all its attributes. The “ProcessModule” class implements “Variantable”, hence it inherits all attributes from “Variantable” AND “Named”.

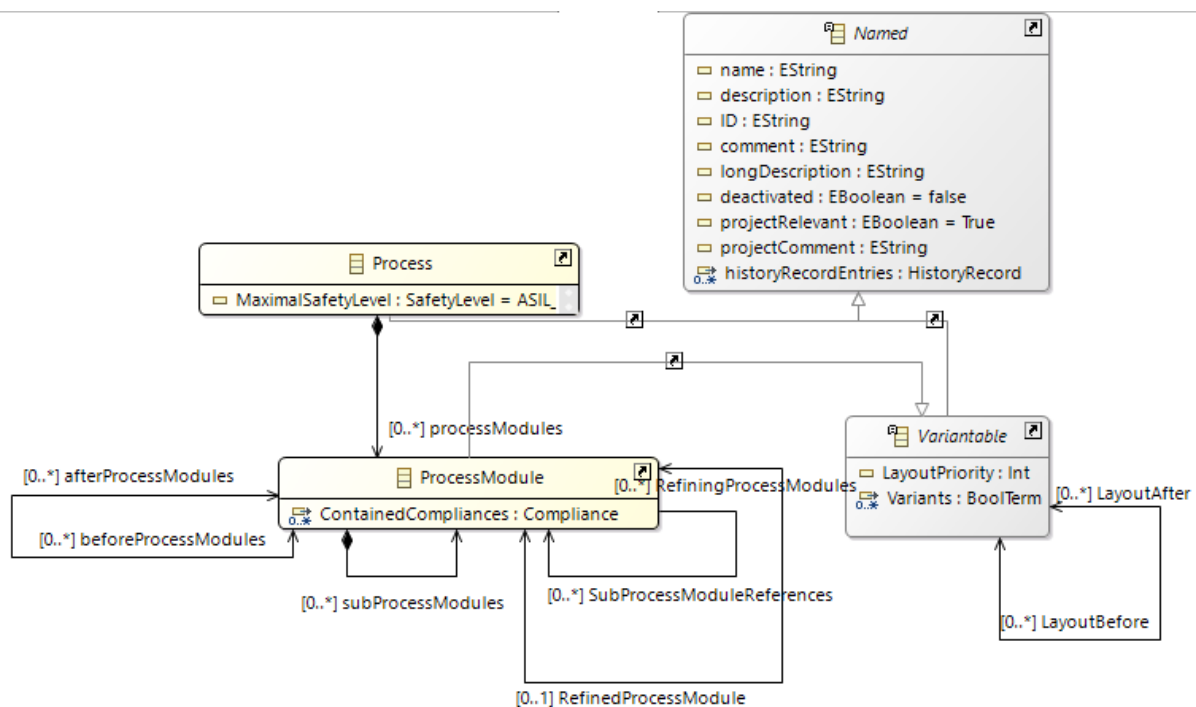


Figure 137: Interfaces in PMT (with examples)

9.3.1 Named

The interface “Named” is a superclass of all named elements (all PMT elements except MetaModel, Binding and Term-Elements have names), allowing them to have the following attributes:

- Attributes:
 - **Name: String:** the name of the element
 - **Description: String:** a short description of the element
 - **LongDescription: String:** a multi-line description of the element
 - **ID: String:** a unique identified for the element

⁶ The interfaces IVerifies and IVerifier (in the implementation) are depreciated and might be removed in future version of PMT. Therefore they are not described here further.

- **Comment: String:** a comment explain the element description
- **Deactivated: Boolean:** if specified this element will be ignored
- **projectRelevant: Boolean (Default=true):** can be used to specify project relevance, e.g. for exporting project specific documents like DIA or offers.
- **projectComment: String:** can be used to explain project specific things, especially if things are not project relevant.

9.3.2 Variantable -> Named

The interface "Variantable" is a superclass of all elements that can have variants (see Section 8.2.6) and graphical layout (see Section 8.2.9), allowing them to have the following attributes (in addition to all properties inherited from the superclass Named):

- Superclass: **Named**, see Section 9.3.1.
- Subclasses:-
- Instances: The following elements are "variantable"
 - **ProcessModule** (and **VerificationModule**)
 - **Artifact** (and **Model**)
 - **StakeHolder**
 - **Criterion**
 - **Tool**
- Attribute:
 - **LayoutPriority: int:** The priority of this element for layout computation: The higher the priority value, the higher will be the priority, i.e. the elements are sorted "descending", see Section 8.2.9 for more details about layout.
- Composition:
 - **Variants: BoolTerm [0..*]:** Terms under which the element is activated. I.e. if one of the contained terms evaluates to true the element is present ("OR-semantic")
- Associations:
 - **LayoutBefore: Variantable [0..*]:** The elements that shall be layouted before this element (by drawing invisible lines from them)
 - **LayoutAfter: Variantable [0..*]:** The elements that shall be layouted after this element (by drawing invisible lines to them)

9.3.3 Verification Interface

The verification interface has been introduced to implement verification relations in a generic way and to make PMT easily extensible for future verification constructs. Currently it is however only used with the Verification Module and the artifacts that are verified. The verification interface (see Figure 138) consists of two abstract classes:

- IVerifier: the verifier
- IVerifiedBy: the verified element

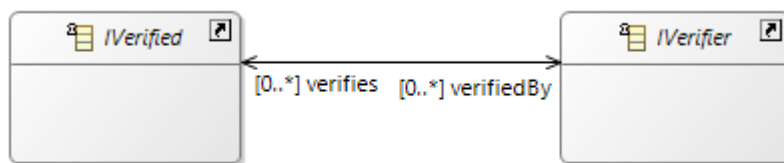


Figure 138: Verification Interface

The interface “IVerified” indicates that the implementing element is verified. Usually artifacts are verified.

- Superclass: -
- Instances: The following elements are “IVerified”
 - ProcessModule (and VerificationModule)
 - Artifact (and Model)
 - Requirement
- Attribute: -
- Composition:
- Associations:
 - **verifiedBy: IVerifier [0..*]:** The verifier (typically verification modules) that verify the element.

The interface “IVerifier” indicates that the implementing element verifies something. Usually VerificationModules are verifiers.

- Superclass: -
- Instances: The following elements are “IVerifier”
 - ProcessModule (and VerificationModule)
- Attribute: -
- Composition:
- Associations:
 - **verifies: IVerifiedBy [0..*]:** The verifier (typically verification modules) that verify the element.

9.4 Scoping, Hierarchy and Reuse

The PMT model is structured as a tree. Most properties are specified “locally” in the elements that have them, for example the name of the element or its description.

Some attributes however are “inherited” into the contained elements, for example if an element is deactivated, it automatically deactivates all its children, even those elements are not specified to be deactivated.

The same holds for the owner of a process module or artifact. If the process module has no owner specified it is associated to the owner of the containing process module. This greatly simplifies specification of responsibilities.

The same holds for variables & parameters. If a variable/parameter shall be evaluated (for example to determine if a variant is true/false), PMT searches for its value first locally in the using element. If it is not found (bound) there the value is searched in the containing element until the value is found or the root element (Container) has been searched for it.

Note that only non-deactivated elements are considered (also in the search for variables).

Most elements (Types, Stakeholder, Artifacts and ProcessModules) can be contained in the global root container (“Process”) or in any other process. For modularity it is better not to store them globally, but local in the process where the elements belong to.

Since the model is a tree structure it is hard to re-use elements (without copy & pasting them, which would make them hard to modify). Therefore we have added so called **References** to the meta model that support the reuse of hierarchic elements without copying it. Technically they are implemented as associations, semantically they are treated like compositions in the generated artifacts (but currently not within the tree browser⁷).

For example there may be many documents, e.g. TQR and a TSM that each has a table of contents consisting of references to sections. Instead of modeling the table of contents twice, we can use references to it in the modeled documents and for sake of reuse put the table of contents into something that we model as template-library.

In the Artifact example the name of the reference is “SubArtifactReferences”. It can be specified as shown in Figure 139 by adding the “Table Of Contents” Artifact from the “Validas Library” Process

⁷ This might change in future, once the feature request #XX is implemented.

as “SubArtifactReference” to the artifact “TQR”, such that the TQR has then not only three children (as shown in the tree browser) but four (including the table of contents) as shown in the Process Module View.

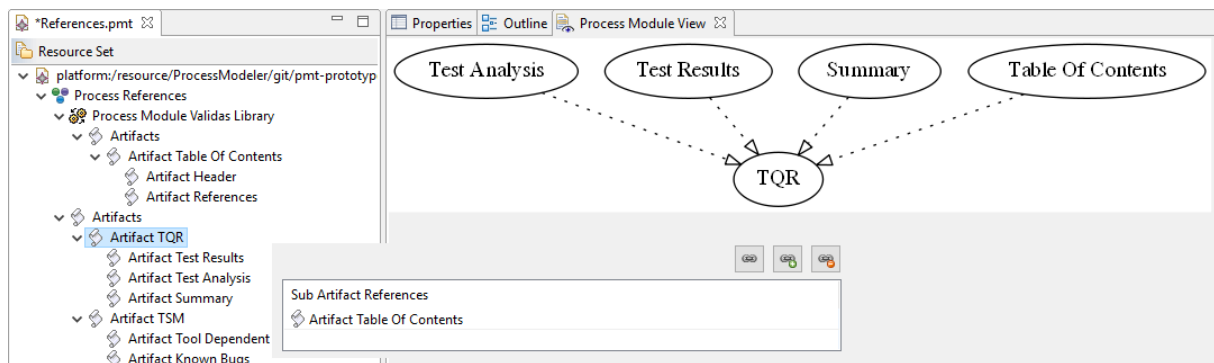


Figure 139: References in Artifacts

The same concept of references can be used with process modules, see Figure 140. Also parameters can be referenced from other modules. In requirements there are also references, but this is called (due to the nature of safety standards that frequently use these pointers) “RequiredRequirements”.

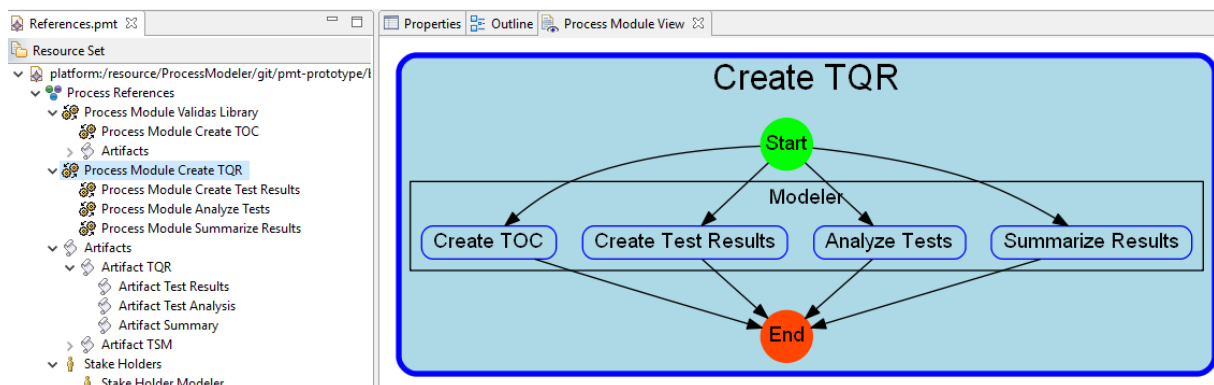


Figure 140: References in Process Modules

9.5 Types

PMT has a static type system. Terms and types build the basis for PMT models and their automated evaluation, see Section 8.1. In this section all modeling elements are described that can be used to express Variants, Conditions and values for Parameters:

- Types: Define the basis of the PMT terms.
- Terms: Define the allowed terms that can be used within PMT.
- Bindings: Bind Values to Parameters.

Parameters are defined in Section 9.8.

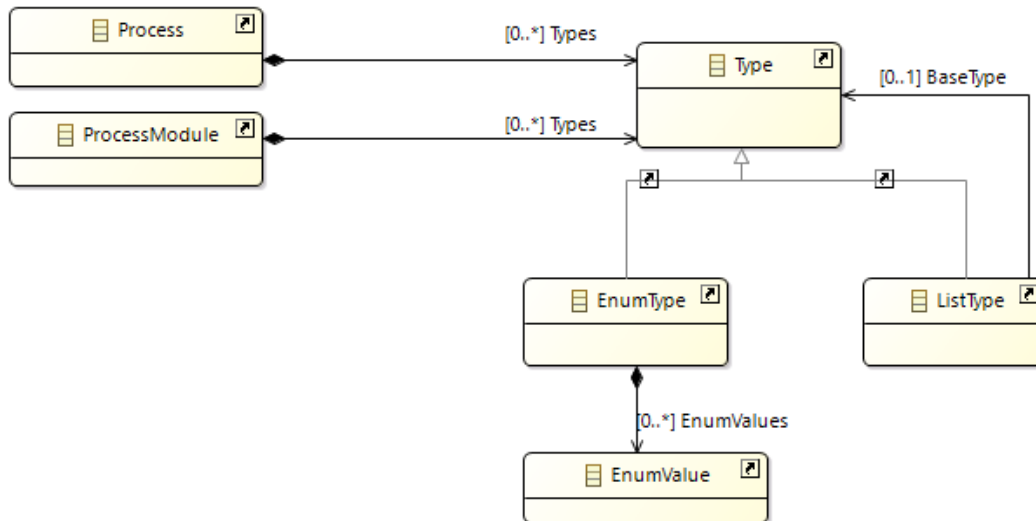


Figure 141: Meta-Model for all Types

Types can be defined in “Process” and “ProcessModule” elements, see Section 8.1. Figure 141 gives an overview on the possible definitions of types.

9.5.1 Type -> Named

The type describes the set of possible values that a Term with that type can have, e.g. a Boolean variable can have the values of type Bool (True and False).

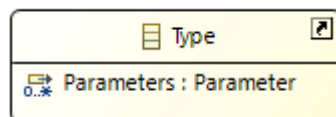


Figure 142: Meta-Model of Type

Type implements Named and in addition it contains the following properties (see Figure 142):

- Superclass: “Named”, see Section 9.3.1.
- SubClasses: **EnumType** and **ListType**
- Container: The following elements can contain Types
 - **Process**
 - **ProcessModule** (and **VerificationModule**)
- Attributes: No additional attributes for Types.
- Associations:
 - Parameters: The parameters that have this type.
- Compositions: No compositions of Type.

9.5.2 EnumType -> Type

EnumType is the type of elements that can have only values from an enumeration that defines the Enumerated Type. The enumerated values ("EnumValue" are contained in the definition of the EnumType).

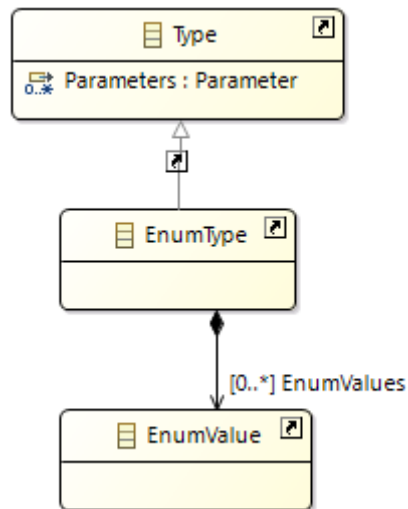


Figure 143: Meta-Model of EnumType with EnumValues

EnumType subclasses Type and in addition it contains the following properties (see Figure 143):

- Superclass: "Type", see Section 9.5.1.
- Container: The following elements can contain Types
 - **Process**
 - **ProcessModule** (and **VerificationModule**)
- Attributes: -
- Associations: -
- Compositions:
 - EnumValue: the enumerated values of the definition.

In contrast to classical programming languages there are no textual definitions of enumerated types (as in Java) that can be used in PMT yet.

9.5.3 EnumValue -> Named

EnumValue represents a single enumerated value of an EnumType. EnumValues are contained in the definition of their types (EnumType)

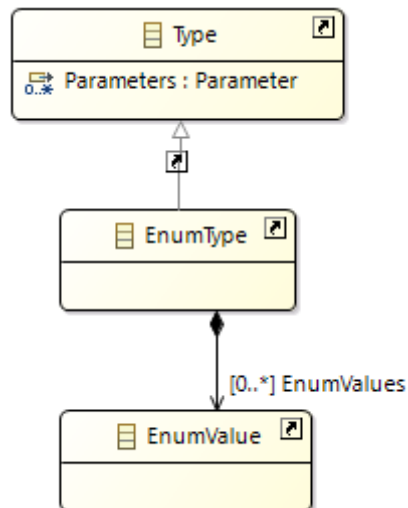


Figure 144: Meta-Model of EnumValue

EnumValue subclasses Named and in addition it contains the following properties (see Figure 144):

- Superclass: "Named", see Section 9.3.1.
- Container: The following element can contain EnumValues:
 - **EnumType**
- Attributes: No additional attributes for EnumValues.
- Associations: No additional associations for EnumValues.
- Compositions: No additional compositions for EnumValues.

9.5.4 ListType -> Type

ListType is the type of lists over values. The contained values in the list have all the same "base type". List terms can be constructed using the ListTerm constructor.

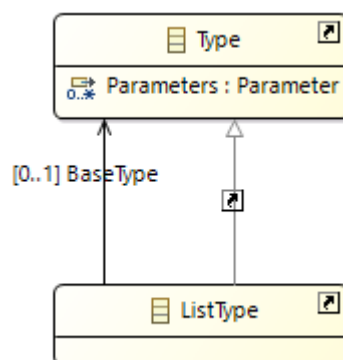


Figure 145: Meta-Model of ListType with Base Type

ListType subclasses Type and in addition it contains the following properties (see Figure 145):

- Superclass: "Type", see Section 9.5.1.

- Container: The following elements can contain ListTypes:
 - **Process**
 - **ProcessModule** (and **VerificationModule**)
- Attributes: -
- Associations:
 - BaseType: Type [0..1]: The base type of the list, e.g. String from ListOfString.
- Compositions: -

In contrast to classical programming languages there are no textual definitions of list types (as in Java) that can be used in PMT yet.

9.6 Terms

Terms describe conditions and values in the PMT model. Boolean terms can be expressed for automated tailorings in the process model. Basically Terms are built, as usual in lambda calculus over the following elements that are described within this section:

- Constants (and enumerated values)
- Operators, e.g. &&,||,==
- Variables, modeled as Parameters described in Section 9.8.

In order to maintain the consistency of the model, we cannot add constants and variables several times into terms. Otherwise we would have duplicated constants denoting the same terms. Therefore we added references to parameters and enumeration values to the term model. Constants are always different, when evaluating them, see Section 8.1. Currently we do not see the need for arithmetic operators within process modeling, therefore we have concentrated mainly on the Boolean terms.

The model of terms is depicted in Figure 146.

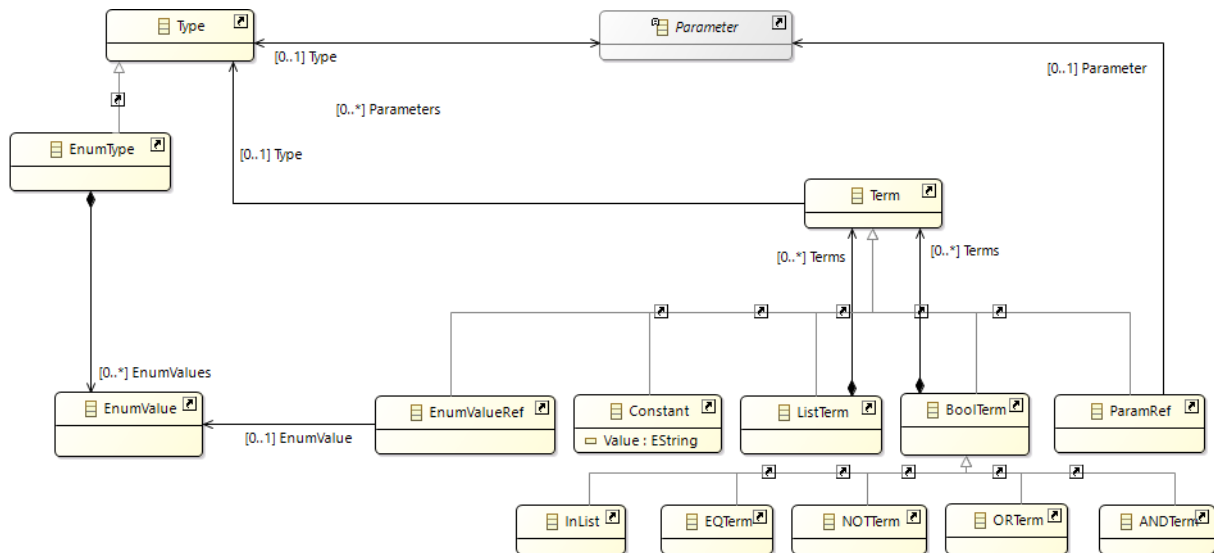


Figure 146: Meta-Model of all Terms

9.6.1 Term

The class Term is an abstract class for all different terms. Every term can have a Type describing its allowed values.

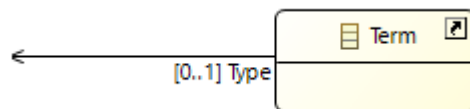


Figure 147: Meta-Model of Term

Term has no Superclass for efficiency reasons (to keep terms small) (see Figure 147):

- Superclass: -
- SubClasses
 - **EnumValueRef**: References to EnumValues.
 - **Constant**: Constant value, e.g. 1 or True.
 - **ListTerm**: a list term, with term arguments (of the same type).
 - **BoolTerm**: Boolean terms with term arguments (of boolean type).
 - **ParamRef**: References to a Parameter.
- Container: The following elements can contain Terms
 - **Variantable** elements:
 - **ProcessModule** (and **VerificationModule**)
 - **Artifact** (and **Model**)
 - **StakeHolder**
 - **Criterion**
 - **Tool**

- **Binding:** contains the bound value (**Term**)
- **ListTerm**
- **BoolTerms:**
 - **ORTerm**
 - **ANDTerm**
 - **EQTerm**
 - **NOTTerm**
 - **InList**
- Attributes: -
- Associations:
 - **Type: Type [0..1]:** the type of a Term
- Compositions: -

9.6.2 EnumValueRef -> Term

An **EnumValueRef** element is a constant term referring to an enumerated value (**EnumValue**, see Section 9.5.3).

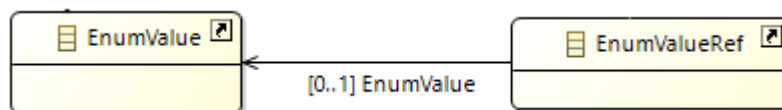


Figure 148: Meta-Model of EnumValueRef

EnumValueRef is a Term allowing to use enumerated values within terms (see Figure 148):

- Superclass: **Term**, see Section 9.6.1.
- SubClasses: -
- Attributes: -
- Associations:
 - **EnumValue: EnumValue [0..1]:** the referred EnumValue.
- Compositions: -

Note for efficiency reasons the relation between EnumValueRef and EnumValue is uni-directional, i.e. the EnumValue does not know which references to it exists.

9.6.3 Constant -> Term

A Constant element is a constant term with a defined and unchangeable value.

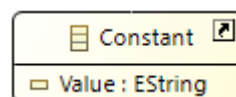


Figure 149: Meta-Model of Constant

Constant terms are used to evaluate terms, see Section 8.1.

- Superclass: Term, see Section 9.6.1.
- SubClasses: -
- Containers: see Term in Section 9.6.1.
- Attributes:
 - Value: String: contains the value of the constant, e.g "1", or "FALSE".
- Associations: -
- Compositions: -

Note that we do not distinguish between different types of constants, e.g. Boolean Constants, Integer Constants,... This is expressed by the assigned Type, which is inherited from the super-class Term.

9.6.4 ListTerm -> Term

A ListTerm element is a Term constructor for building lists (comparable to "[.]") that takes a list of term arguments and creates a list of it.

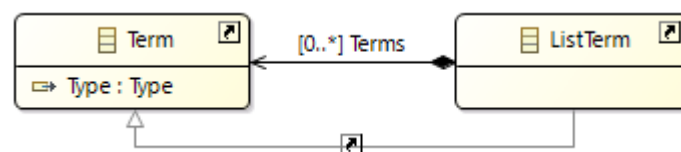


Figure 150: Meta-Model of ListTerm

ListTerm is a Term allowing to use enumerated values within terms (see Figure 148):

- Superclass: Term, see Section 9.6.1.
- SubClasses: -
- Containers: see Term in Section 9.6.1.
- Attributes: -
- Associations: -
- Compositions:
 - Terms: Term [0..*]: List of arguments/elements in the list.

9.6.5 BoolTerm -> Term

A BoolTerm element is a Term constructor for building Boolean terms (comparable to operators like &&, ||, ==,..) that takes a list of term arguments and creates Boolean expressions of them, for examples "TRUE && X".

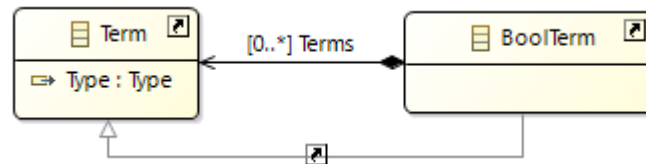


Figure 151: Meta-Model of BoolTerm

BoolTerm is a Term with a boolean result that can be used in conditions (see Figure 151):

- Superclass: Term, see Section 9.6.1.
- SubClasses:
 - InList: checks if an element is in a list.
 - ANDTerm: is true if both arguments are true.
 - ORTerm: is true if one argument is true.
 - NOTTerm: is true if the argument is false.
 - EQTerm: is true if the terms are equal.
- Containers: see Term in Section 9.6.1
- Attributes: -
- Associations: -
- Compositions:
 - Terms: Term [0..*]: List of arguments of the BoolTerm.

9.6.6 ParamRef -> Term

A ParamRef element is a constant term referring to a parameter (Parameter, see Section 9.8).

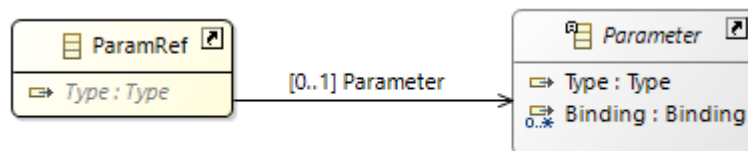


Figure 152: Meta-Model of EnumValueRef

ParamRef is a Term allowing to use parameters values within terms (see Figure 152):

- Superclass: Term, see Section 9.6.1.
- SubClasses: -
- Containers: see Term in Section 9.6.1.
- Attributes: -
- Associations:
 - Parameter: Parameter [0..1]: the referred Parameter.
- Compositions: -

9.6.7 InList -> BoolTerm

An InList element is a boolean term for checking if an element is in a list of elements. Of course the types have to be compliant, i.e. a String can only be in a list of Strings.

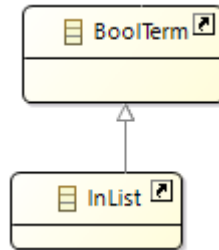


Figure 153: Meta-Model of InList

InList is a special boolean term (see Figure 153) with the following properties:

- Superclass: BoolTerm, see Section 9.6.5.
- SubClasses: -
- Containers: see Term in Section 9.6.1.
- Attributes: -
- Associations: -
- Compositions: -

9.6.8 ORTerm -> BoolTerm

An ORTerm element is a boolean term for expressing a logical “or” operation, e.g. A || B.

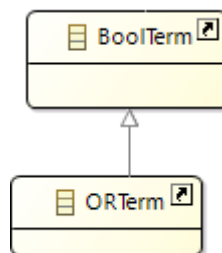


Figure 154: Meta-Model of ORTerm

ORTerm is a special boolean term (see Figure 154) with the following properties:

- Superclass: BoolTerm, see Section 9.6.5.
- SubClasses: -
- Containers: see Term in Section 9.6.1.
- Attributes: -
- Associations: -
- Compositions: -

9.6.9 ANDTerm -> BoolTerm

An ANDTerm element is a boolean term for expressing a logical “and” operation, e.g. A && B.

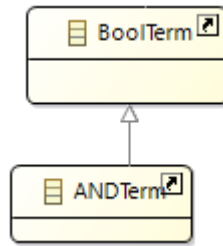


Figure 155: Meta-Model of ANDTerm

ANDTerm is a special boolean term (see Figure 155) with the following properties:

- Superclass: BoolTerm, see Section 9.6.5.
- SubClasses: -
- Containers: see Term in Section 9.6.1.
- Attributes: -
- Associations: -
- Compositions: -

9.6.10 NOTTerm -> BoolTerm

An NOTTerm element is a boolean term for expressing a logical negation, for example !A.

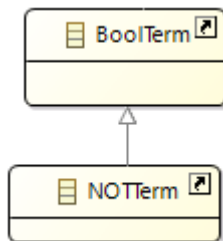


Figure 156: Meta-Model of NOTTerm

NOTTerm is a special boolean term (see Figure 155) with the following properties:

- Superclass: BoolTerm, see Section 9.6.5.
- SubClasses: -
- Containers: see Term in Section 9.6.1.
- Attributes: -
- Associations: -
- Compositions: -

9.6.11 EQTerm -> BoolTerm

An EQTerm element is a boolean term for expressing equality, for example $A=B$.

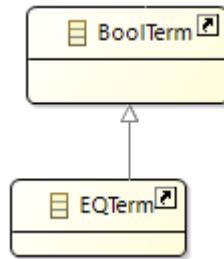


Figure 157: Meta-Model of EQTerm

EQTerm is a special boolean term (see Figure 155) with the following properties:

- Superclass: BoolTerm, see Section 9.6.5.
- SubClasses: -
- Containers: see Term in Section 9.6.1.
- Attributes: -
- Associations: -
- Compositions: -

9.7 Bindings

Bindings allow to bind variables in terms to values. Variables in terms are parameters of the process. Bindings contain a Term that is used for the bound variables. Bindings can be declared locally (within ProcessModules), or globally with Process elements.

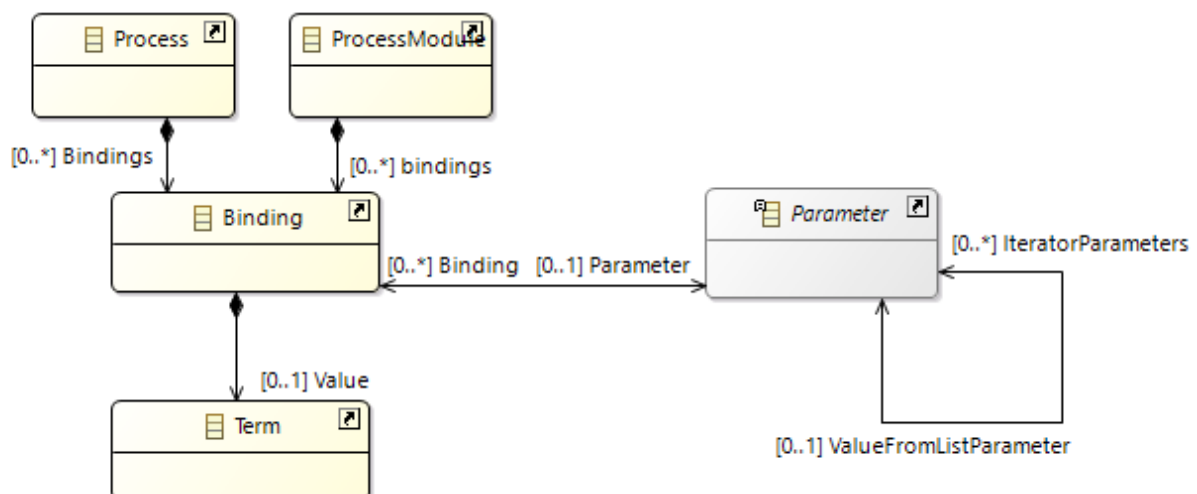


Figure 158: Meta-Model of Bindings

Binding is a modeling (see Figure 158) with the following properties:

- Superclass:

- SubClasses: -
- Container: The following elements can contain Bindings:
 - **Process**
 - **ProcessModule** (and **VerificationModule**)
- Attributes: -
- Associations: -
 - Parameter: Parameter [0..1]: The parameter to which the value is bound.
- Compositions:
 - Value: Term [0..1]: the term to which the Parameter is bound (after evaluating it), see Section 8.1.

Note that the search for parameter bindings is not done using the association in the model, but according to the scope in which the term is evaluated. So if the parameter X is evaluated in a process P, than the Bindings of the process P are checked if they bind the parameter X. If not the parents of P,.. until the global bindings in Process are considered.

9.8 Parameters

Parameters can be used to tailor and instantiate processes, see Sections 8.2.6 and 8.2.7. A parameter of a process is something that can change its value and have impact of the process, for example the selected safety standard or the name of a tool.

There are different kinds of parameters, that can be used to express different intentions of the parameter:

- Process Parameter: A parameter that impacts the process (before starting the process), e.g. the relevant safety standard, or the type of the qualified object (Tool / Library / Software).
- Planning Parameter: A parameter used for planning the projects & efforts, e.g. the number of tools or lines of code.
- Project Parameter: A parameter that is determined during the project, e.g. the name of the modeled feature or tool.
- Process Variable: a parameter that impacts the process that is determined within the project, e.g. the criticality of a tool. This is somehow a combination of ProcessParameter and a project parameter. ProcessVariables can be used to model process decisions, see Section 8.2.8.

The Process, Planning and Project parameters have currently no semantic differences in PMT, so they can be chosen based on the intuition.

Parameters are always parameters of ProcessModules, so a ProcessModule is like a function that has parameters and can be reused for different values of the parameters, including a parameter dependent behavior.

Since parameters shall be re-usable by many process modules they can also be referred from ProcessModules using the association "ParameterReferences". This reference mechanism allows also to declare parameters globally (in Process) and reuse them in all ProcessModules that shall have them.

Note: if a ProcessModule has a parameter, this implies that automatically all contained process modules "inherit" the same parameter.

Parameters are modeled as described in Figure 159.

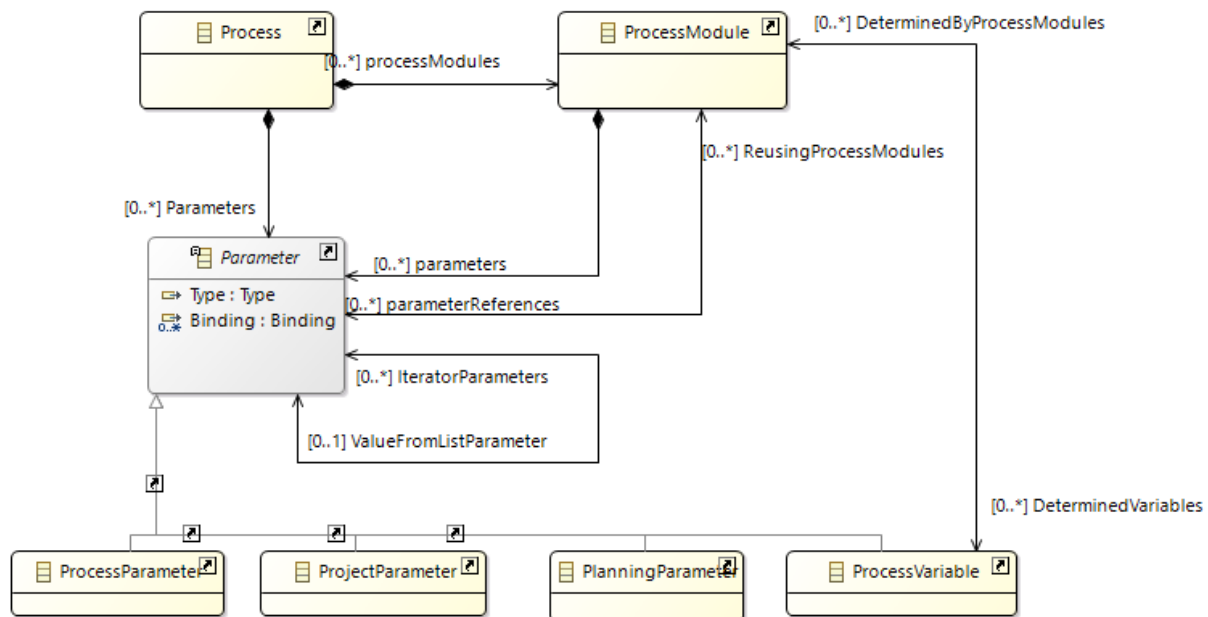


Figure 159: Meta Model of Parameters

9.8.1 Parameter -> Named

Parameter is an abstract class for all PMT parameters.

Parameter has the following properties:

- Superclass: Named, see Section 9.3.1.
- SubClasses:
 - ProcessParameter
 - PlanningParameter
 - ProjectParameter
 - ProcessVariable
- Container: The following elements can contain Parameters
 - **Process**
- Attributes: -
- Associations:
 - **Type: Type [0..1]**: The type of the parameter.
 - **ReusingProcessModules: ProcessModules [0..*]**: ProcessModules that reuse the parameter.

- **IteratorParameters:Parameter [0..*]**: The parameters that iterate over this parameter (only meaningful if this parameter is a list parameter), see Section 8.2.8 for the concept of iterators.
- **ValueFromListParameter: Parameter [0..1]**: The parameter (list), where this parameter receives its values from, see Section 8.2.8 for the concept of iterators.
- Compositions: -

9.8.2 ProcessParameter -> Parameter

A Process Parameter is a specific Parameter. Its values are determined during process compliance phase before planning the project. ProcessParameter has the following properties:

- Superclass: Parameter, see Section 9.8.1.
- SubClasses: -
- Container: The following elements can contain all kinds of Parameters
 - **ProcessModule** (and **VerificationModule**)
- Attributes: -
- Associations: -
- Compositions: -

Note: semantically Project-, Planning- and Process-Parameters are equal.

9.8.3 PlanningParameter -> Parameter

A Planning Parameter is a specific Parameter. Its values are determined when planning the project, e.g. during offer creation phase.

PlanningParameter has the following properties:

- Superclass: Parameter, see Section 9.8.1.
- SubClasses: -
- Container: The following elements can contain all kinds of Parameters
 - **ProcessModule** (and **VerificationModule**)
- Attributes: -
- Associations: -
- Compositions: -

Note: semantically Project-, Planning- and Process-Parameters are equal.

9.8.4 ProjectParameter -> Parameter

A Project Parameter is a specific Parameter. Its values are determined within the project. **ProjectParameter** has the following properties:

- Superclass: Parameter, see Section 9.8.1.
- SubClasses: -
- Container: The following elements can contain all kinds of Parameters
 - **ProcessModule** (and **VerificationModule**)

- Attributes: -
- Associations: -
- Compositions: -

Note: semantically Project-, Planning- and Process-Parameters are equal.

9.8.5 ProcessVariable -> Parameter

A Process Variable is a specific Parameter. Its values are determined within the project but also impact the process. For example a TestResult could have the values true/false and trigger different sub-processes.

ProcessVariable has the following properties:

- Superclass: Parameter, see Section 9.8.1.
- SubClasses: -
- Container: The following elements can contain all kinds of Parameters
 - **ProcessModule** (and **VerificationModule**)
- Attributes: -
- Associations:
 - DeterminedByProcessModules: ProcessModule [0..*]: the process modules that determine the value of the variable, e.g. the ProcessModule "Run Test" determines the variables of the ProcessVariable TestResult, see Section 8.2.8.
- Compositions: -

Note: semantically Project-, Planning- and Process-Parameters are equal but ProcessVariables have a different behaviour when drawing processes graphically.

9.9 Process Frame

Main element of the process model are processes and process modules, see Figure 160.

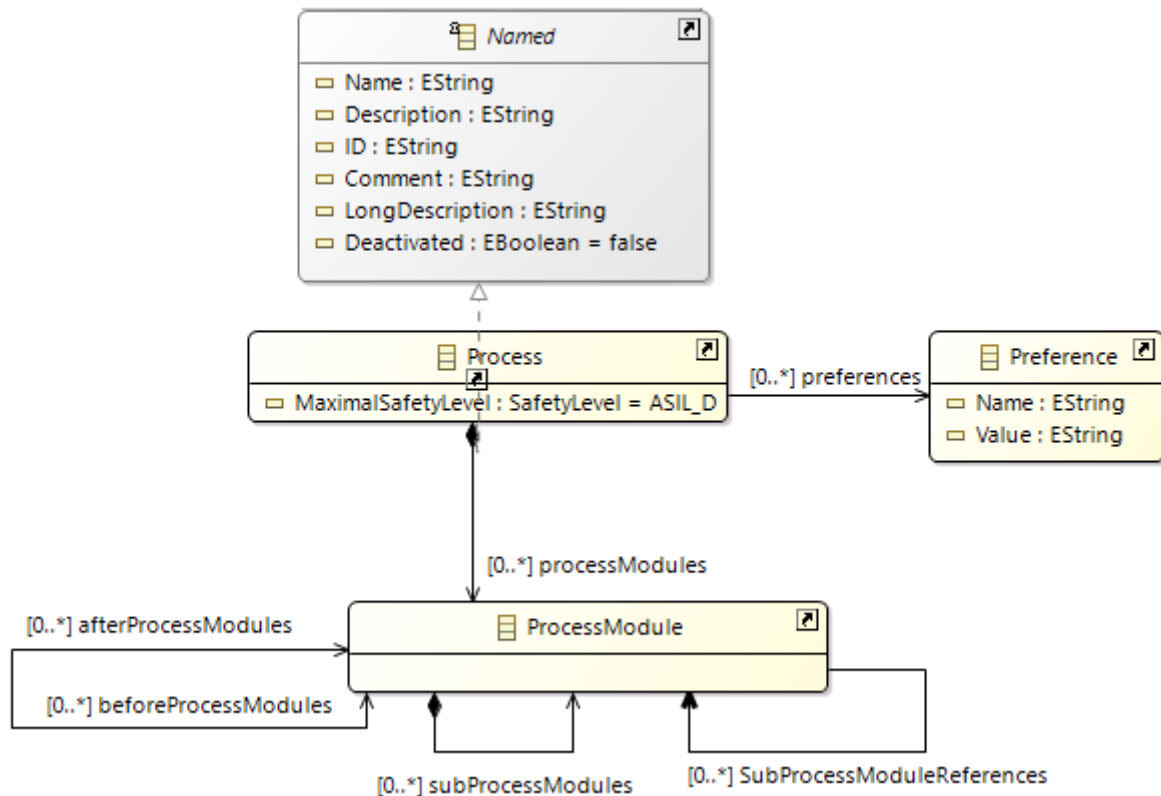


Figure 160: Processes

9.9.1 Process -> Named

The element **Process** is the root element for all process models in PMT. See Section 6 for the creation of new models using **Process** elements.

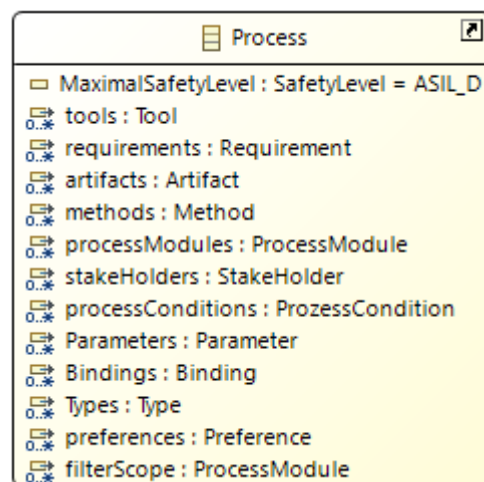


Figure 161: Meta-Model of Process with all Attributes and References

Process implements **Named** and in addition it contains the following properties (see Figure 161):

- **Superclass**: "Named", see Section 9.3.1.

- Attributes:
 - **MaximalSafetyLevel: SafetyLevel:** Specifies the maximal safety level of the process. ASIL_D is the default value:
- Associations:
 - **FilterScope: ProcessModule: [0..*]:** the ProcessModules that shall be used in selection dialogs (to filter not relevant elements).
- Containments:
 - **Tools: Tool [0..*]:** The tools used globally within this process.
 - **Requirements: Requirement[0..*]:** The available requirements within this process.
 - **Artifacts: Artifact [0..*]:** The (global) artifacts available in the process.
 - **Methods: Method [0..*]:** the available methods in the process.
 - **ProcessModules: ProcessModule [0..*]:** The available processModules in the process.
 - **Stakeholders: Stakeholder [0..*]:** The available stake holders / roles in the process.
 - **Deprecated⁸: ProcessConditions: ProcessCondition [0..*]:** The available conditions in the process.
 - **Bindings: Binding [0..*]:** The bindings of variables and parameters in the process.
 - **Parameters: Parameter [0..*]:** The (global) parameters in the process.
 - **Types: Type [0..*]:** The (global) type definitions in the process.
 - **ToBeImplemented⁹: Preferences:Preference [0..*]:** Preferences would allow to store the model relevant preferences, e.g. validation rule settings in the mode, such that models would validate identical in all environments, independent from the local PMT preferences. Also the filter-scopes could be stored as preferences.
 - **FilterScope: ProcessModule: [0..*]:** the ProcessModules that shall be used in selection dialogs (to filter not relevant elements). This is just a useful simplification of editing in case many processes are stored within one model.

⁸ This is a feature from the research project SPEDIT and might be removed in future versions of PMT.

⁹ This is a feature that is already implemented in the meta model, but not supported from the rest of the tool, such that it does not work as described.

9.9.2 ToBeImplemented: Preference

The element Preference is an element in the model to store Preferences in the model. Usually Eclipse preferences are user specific, but some PMT preferences should be specified in the model to be equal for all users of the model.

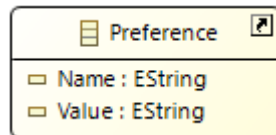


Figure 162: Meta-Model of Preference with all Attributes

Preference contains the following properties (see Figure 162):

- Attributes:
 - **Name: String:** Specifies the name of the preference.
 - **Value: String:** Specifies the value of the preference.

Note: Preferences can be fixed within the project and also stored using the default Eclipse mechanisms.

9.9.3 History Records

History records can be used to document the history of the model. This is especially required, since the generated documents have all the same version 0.8 or 1.0.

History records are contained in the global Process container and have an association to named, such that it is possible to specify the changes by assigning the changed elements to the history record.

In the generated document all history records are listed that are

- Linked to elements contained in the selected process module
- Linked to process modules containing the selected modules
- Global (unlinked) records

This allows to have several models in one file and do not list all (unrelated changes) to a process module.

History records are displayed as shown in Figure 163.

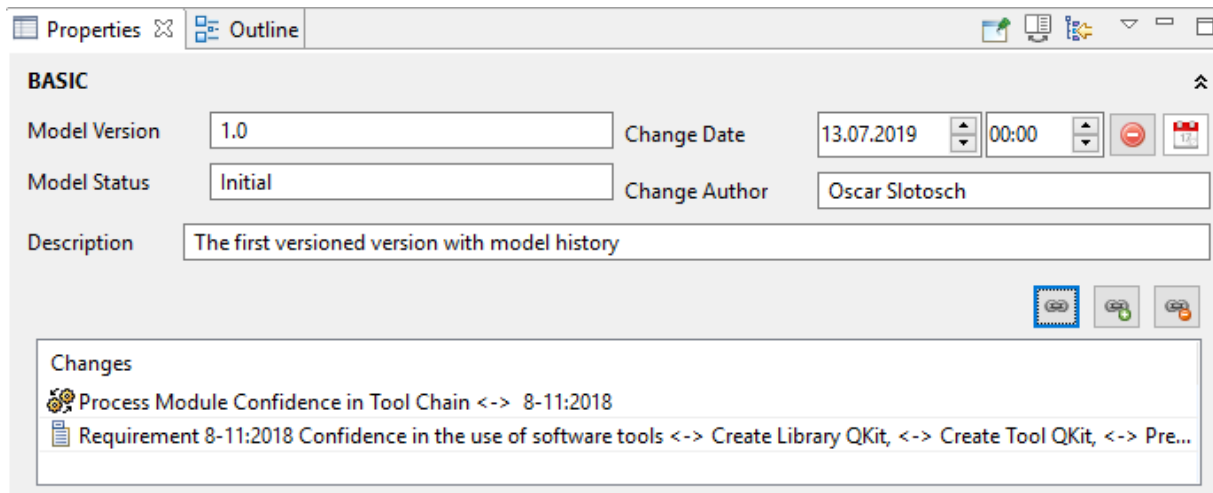


Figure 163: History Record Basic Properties

History Records implements Named and in addition it contains the following properties (see Figure 164):

- Superclass: "Named", see Section 9.3.1.
- Attributes:
 - **changedDate: EDate**: The date of the change (If the time is 00:00 it will not be added to the generated history)
 - **changedAuthor: String**: The name of the author that did the changes
 - **modelStatus: ModelStatus**: The status of the model after the change. It can have the following values:
 - INITIAL (default)
 - DRAFT
 - IN_PROGRESS
 - GENERATED
 - REVIEWED
 - FINAL
 - VALIDATED
 - MODELED
 - VERIFIED
 - RELEASED
 - PRESENTED
 - **modelVersion: String**: The version of the model
- Associations:
 - **Changes: Named: [0..*]**: The changed / affected model elements

Note that the order of the records is determined by the dates, if present, otherwise alphabetical order of the model version strings. If this is not

desired the IDs can be used to determine the order instead of the version strings.

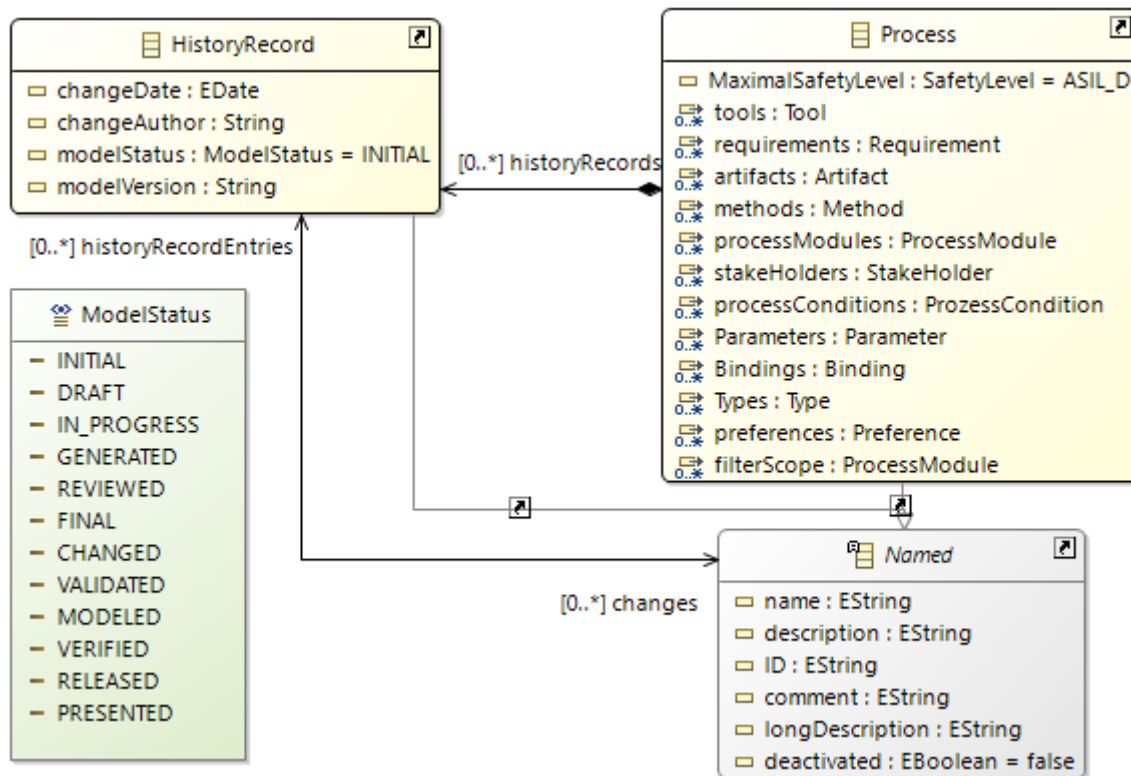


Figure 164: History Records

In the generated reports the history records are displayed in a table (see

Version	Date	Status	Change of model	Author
0.1	Jul 13, 2019	INITIAL	Version 1.0 The first versioned version with model history. Created with current standards (main parts): ISO-26262:2018, IEC-61508 2nd, EN 50657	Oscar Slotosch
0.2	Jul 19, 2019	IN_PROGRESS	New QKit Model Started. Change: <ul style="list-style-type: none"> Qualify Tool with QKit 	Oscar Slotosch
0.3	Jul 21, 2019	IN_PROGRESS	Corrected some typos. Change: <ul style="list-style-type: none"> LibraryQKit 	Oscar Slotosch
0.4	Jul 21, 2019	IN_PROGRESS	Added Tool QKit Process. Change: <ul style="list-style-type: none"> Qualify Tool with QKit 	Oscar Slotosch

Table 135 Model History

Figure 165).

Version	Date	Status	Change of model	Author
0.1	Jul 13, 2019	INITIAL	Version 1.0 The first versioned version with model history. Created with current standards (main parts): ISO-26262:2018, IEC-61508 2nd, EN 50657	Oscar Slotosch
0.2	Jul 19, 2019	IN_PROGRESS	New QKit Model Started. Change: <ul style="list-style-type: none"> Qualify Tool with QKit 	Oscar Slotosch
0.3	Jul 21, 2019	IN_PROGRESS	Corrected some typos. Change: <ul style="list-style-type: none"> LibraryQKit 	Oscar Slotosch
0.4	Jul 21, 2019	IN_PROGRESS	Added Tool QKit Process. Change: <ul style="list-style-type: none"> Qualify Tool with QKit 	Oscar Slotosch

Table 135 Model History

Figure 165: Generated History Record (Report)

A smart way to create history records and assign them to the changed elements is the "Add History Record" Action that works on every Named element. This is done in the following steps:

- 1) Select the changed elements (or their containers) in the tree browser
- 2) Start the "Add History Record" Action as shown in Figure 166
- 3) Confirm the creation as shown in Figure 166
- 4) Update the created element by specific comments

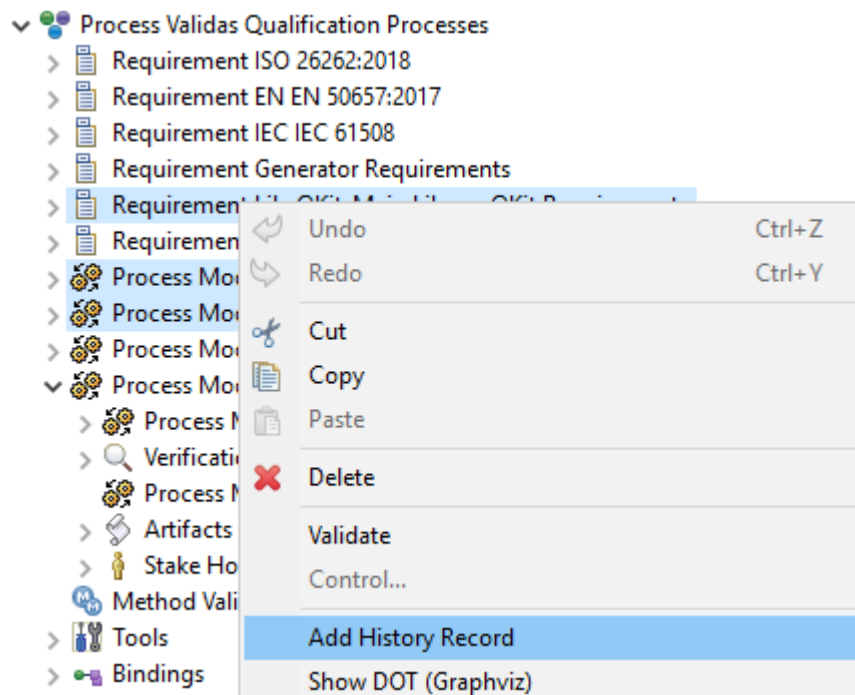


Figure 166: Starting "Add History Record Action"

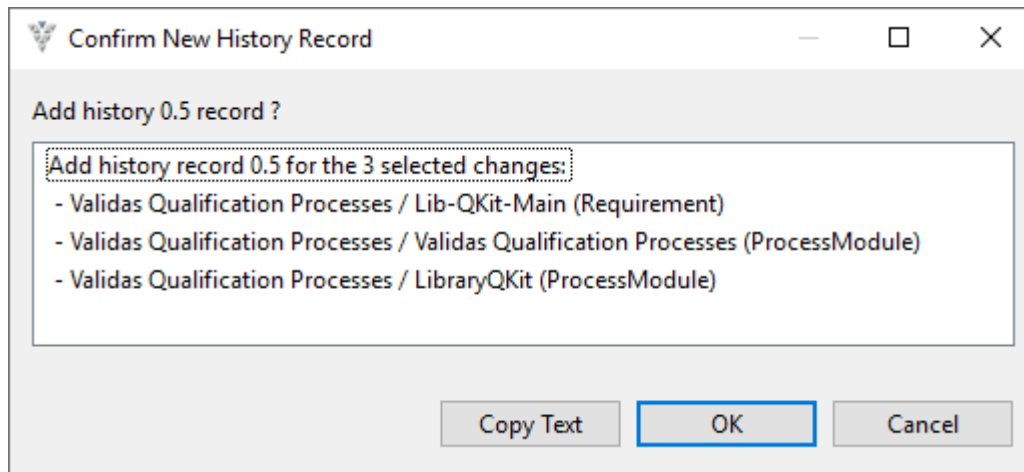


Figure 167: Confirm “New History Record”

The generated history record has the following properties (see Figure 168):

- The new model version: This is computed by incrementing the last digit from the most recent model version by one (starting with 0.1 if there is no previous version found)
- The date of performing the action
- The status: same as previous one (or initial if none was found)
- The author: the name of the current user
- Description: “Changed with PMT”. This should be changed
- Changes: The list of performed changes / selected elements.

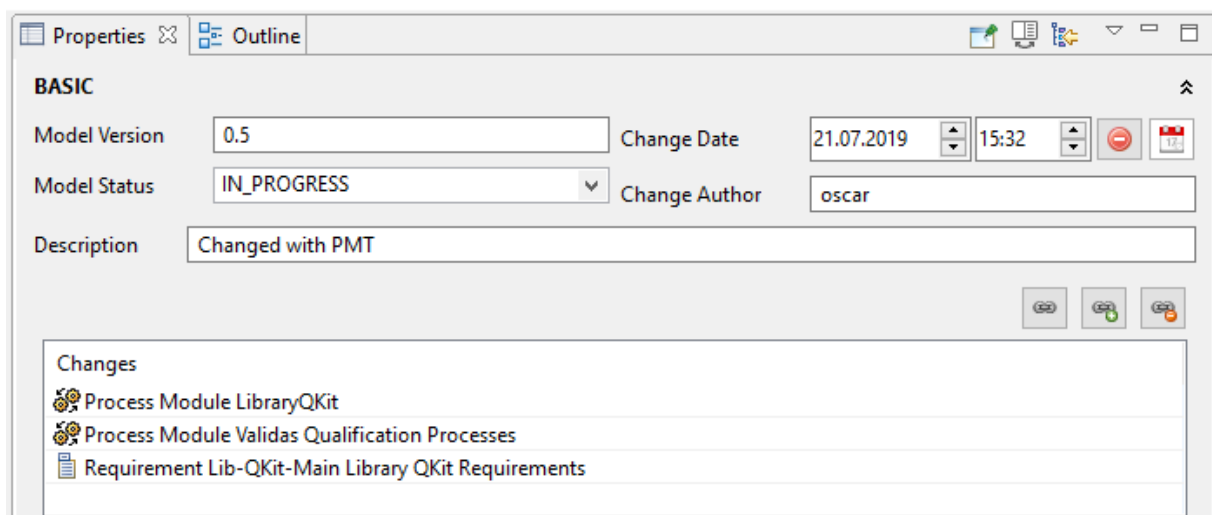


Figure 168: Properties of generated History Record

9.10 Process Models

The core of PMT are process models. Figure 169 gives an overview on the main process elements that are all contained in the Process container or in ProcessModules:

- ProcessModule: main structuring element for processes, represents activities in the processes.
- Artifacts: main data in processes.
- StakeHolder: the acting roles in processes.
- VerificationModule: special verification module to perform verification activities.
- Criterion: Question that has to be answered during verification.

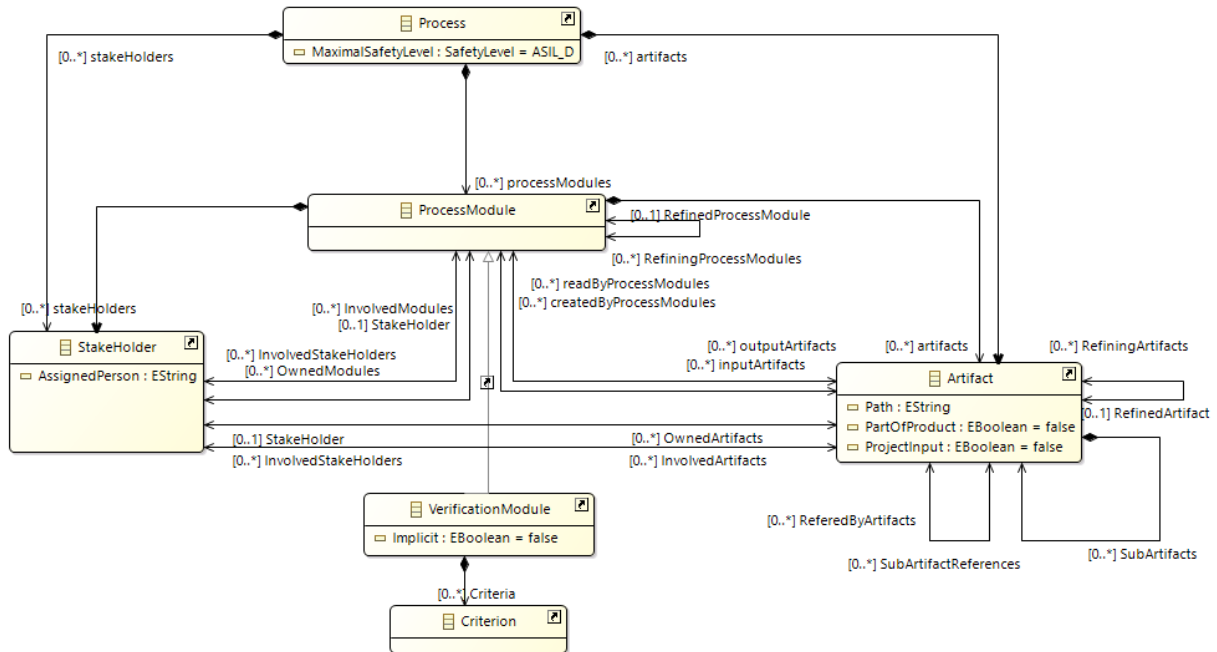


Figure 169: Main Process Elements

9.10.1 Process Module -> Variantable

The element ProcessModule is the main element for modeling processes. It describes activities within the process that are performed by stake holders and produce output artifact by processing input artifacts. They can also be performed before / after other process module.

Process modules can have sub-modules and references to other processes (see Section 8.2.5 for description of reference concept). Since process modules correspond to tasks they can also be used for process management and planning efforts.

ProcessModules can claim to satisfy requirements and they can be used to implement process requirements. Verification Modules are a special form ("subclass") of ProcessModules.

Figure 170 shows the definition of ProcessModule with all references and compositions (modeled as attributes in the diagram).

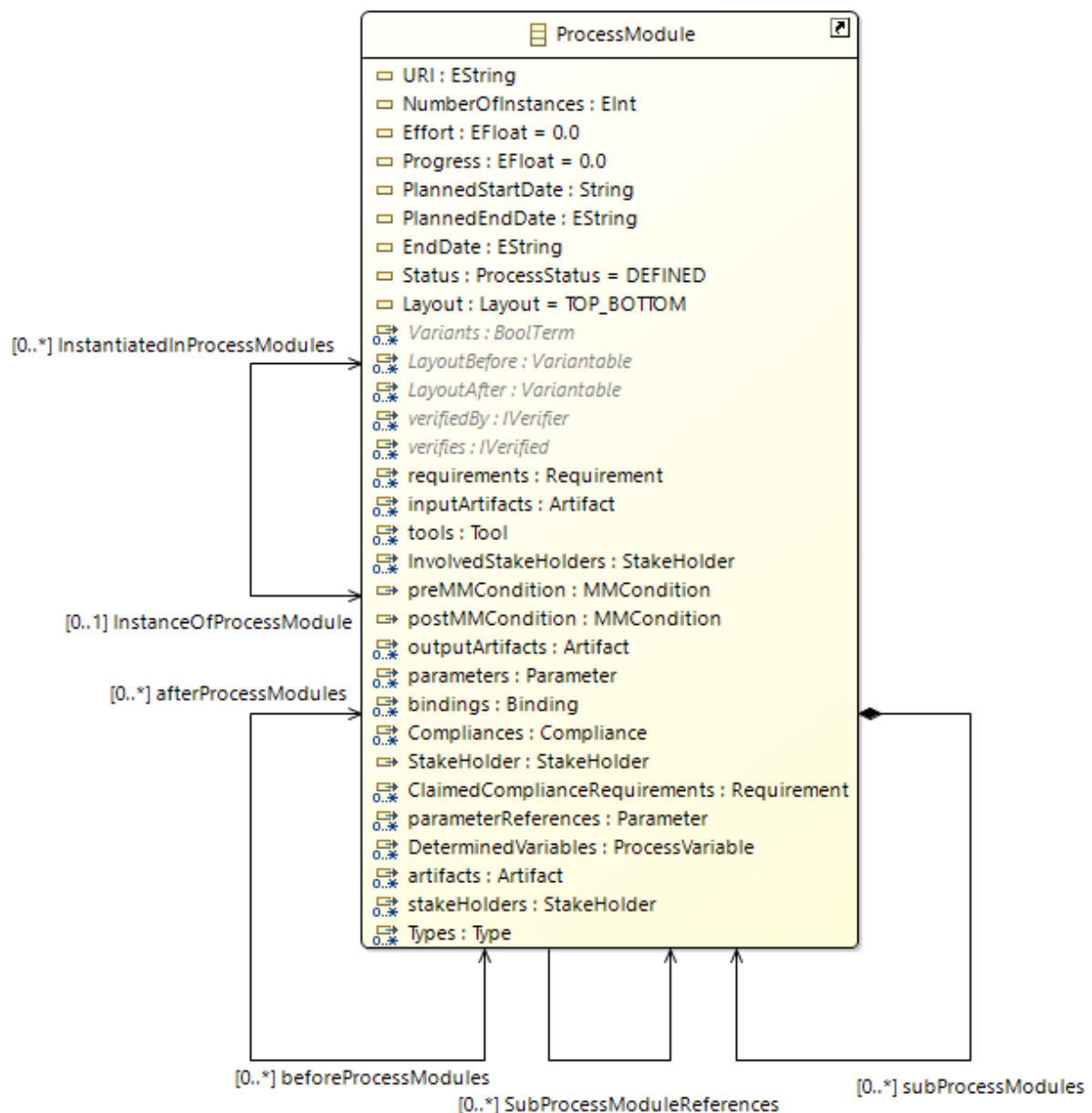


Figure 170: Meta-Model of ProcessModule with all Attributes and References

Therefore the element ProcessModule contains the following properties that can be edited:

- Superclass: **Variantable**, see Section 9.3.2.
- Subclass: **VerificationModule**, see Section 9.10.4.
- Container: The following elements can contain ProcessModules:
 - **Process**
 - **ProcessModule** (and **VerificationModule**)
- Attributes:
 - **URI: String**: an Unique Resource Identifier, pointing to a more detailed process description, e.g. a Wiki-page
 - **NumberOfInstance: int**: The number of planned instances of the process. This is only optional.

- **Effort: float:** The estimated effort for performing this task, that can be used to plan projects and to estimate project status.
- **Progress: float:** can be used to manually trace the progress to determine the project status.
- Associations:
 - **Requirements: Requirement [0..*]:** the (usually atomic) requirements that this process module implements. Note this is not the claimed requirements, they usually do contain/require many other requirements.
 - **ClaimedComplianceRequirements: Requirement [0..*]:** the main requirements that this process module claims to be compliant.
 - **InputArtifacts: Artifact [0..*]:** the input artifacts that are processed by this module.
 - **OutputArtifacts: Artifact [0..*]:** the output artifacts that are created by this process module.
 - **Tools: Tool [0..*]:** The tools that are used within this process module.
 - **RefinedProcessModule: ProcessModule [0..1]:** The process module that is refined by this process module, see Section 8.3
 - **RefiningProcessModules: ProcessModule [0..*]:** The refining process modules that specialize this process module, see Section 8.3.
 - **beforeProcessModules: ProcessModule [0..*]:** the process modules that this module is before, so the related process modules come after this process.
 - **afterProcessModules: ProcessModule [0..*]:** the process modules that this module is after, so the related process modules come before this process.
 - **StakeHolder: Stakeholder [0..1]:** The responsible stakeholder for this process.
 - **InvolvedStakeholders: Stakeholder [0..*]:** Other involved stakeholders (not the responsible one).
 - **Deprecated: preMMCondition: MMCondition [0..1]:** meta model condition that has to be satisfied before the process can be executed (only meaningful within model-based processes).
 - **Deprecated: postMMCondition: MMCondition [0..1]:** meta model condition that has to be satisfied after the process can be executed (only meaningful within model-based processes).
 - **SubProcessModuleReferences: ProcessModule [0..*]:** the process modules that are included via references here, see Section 8.2.5.

- **InstanceOfProcessModule: ProcessModule [0..1]:** The generic, parameterized process module that this process is an instance of. See Section 8.2.7 for instantiations.
- **InstantiatedInProcessModules: ProcessModule[0..*]:** The processes that are instances of this generic process modules. See Section 8.2.7 for instantiations.
- **Compliances: Compliance[0..*]:** The compliances that this process is contributing.
- **ParameterReferences: Parameter[0..*]:** The referred parameters from other modules that also apply to this module. See Section 8.2.5 about parameterization.
- **DeterminedVariables: ProcessVariable[0..*]:** The process variables that values are determined from this process module, see Section 8.2.8 for using process variables.
- Containments:
 - **Tools: Tool [0..*]:** The tools used globally within this process.
 - **Requirements: Requirement[0..*]:** The available requirements within this process.
 - **Artifacts: Artifact [0..*]:** The (global) artifacts available in the process.
 - **Methods: Method [0..*]:** the available methods in the process.
 - **ProcessModules: ProcessModule [0..*]:** The available processModules in the process.
 - **StakeHolders: StakeHolder [0..*]:** The available stake holders / roles in the process.
 - **Deprecated¹⁰: ProcessConditions: ProcessCondition [0..*]:** The available conditions in the process.
 - **Bindings: Binding [0..*]:** The bindings of variables and parameters in the process.
 - **Parameters: Parameter [0..*]:** The (global) parameters in the process.
 - **Types: Type [0..*]:** The (global) type definitions in the process.
 - **ToBeImplemented¹¹: Preferences: Preference [0..*]:** Preferences would allow to store the model relevant preferences, e.g. validation rule settings in the mode, such that models would validate identical in all environments, independent from the local

¹⁰ This is a feature from the research project SPEDIT and might be removed in future versions of PMT.

¹¹ This is a feature that is already implemented in the meta model, but not supported from the rest of the tool, such that it does not work as described.

PMT preferences. Also the filter-scopes could be stored as preferences.

9.10.2 Artifact -> Variantable

Artifacts represent the data in the process, e.g. a specification, model or code. Artifacts can be contained in Process (global artifacts) and they can be contained in process modules (local artifacts). They can be modeled as depicted in Figure 171.

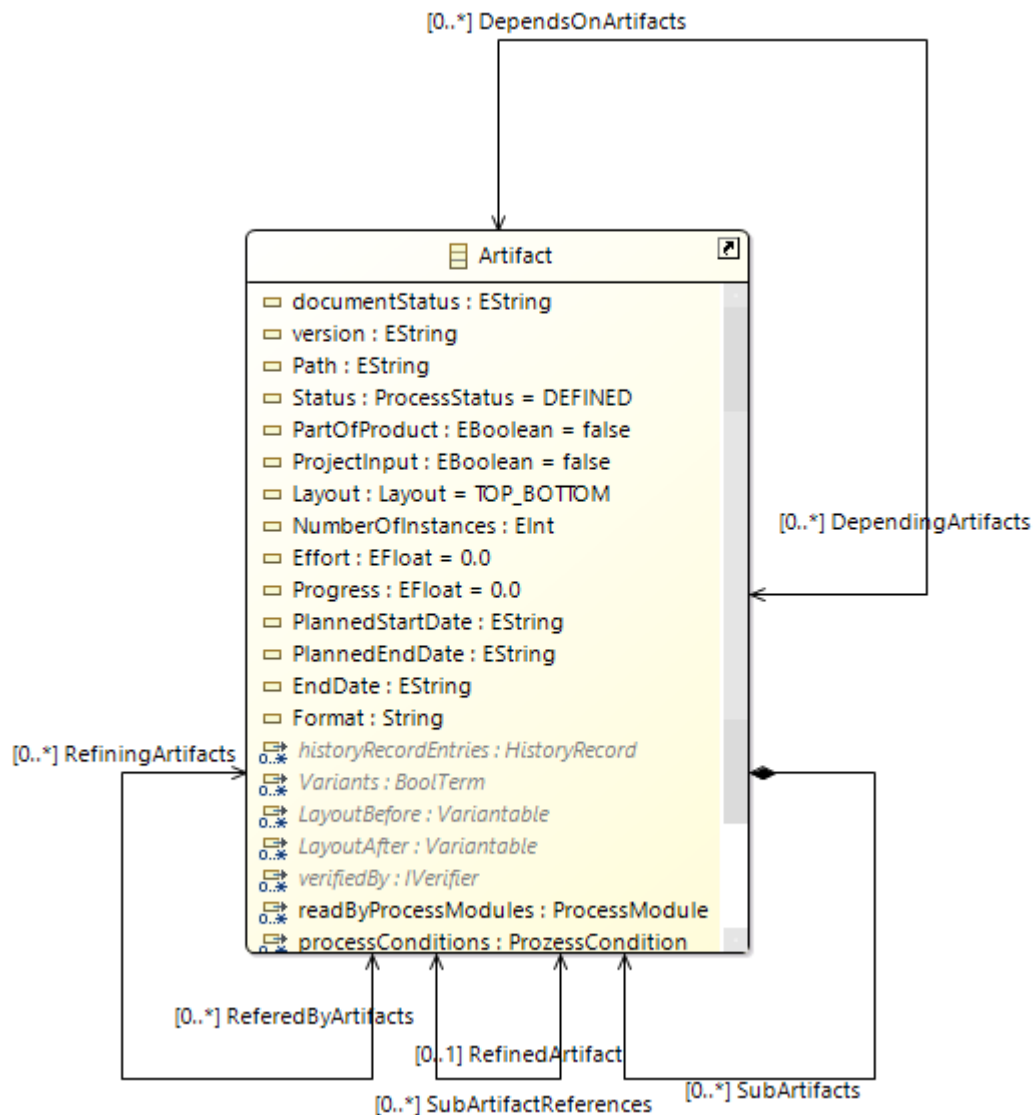


Figure 171: Meta-Model of Artifacts with all Properties

Artifact has the following properties:

- Superclass:
 - **Variantable**, see Section 9.3.2.
 - **IVerified**, see Section 9.3.3.
- SubClasses:
 - **Model**

- Container: The following elements can contain Artifacts:
 - **Process**
 - **ProcessModule** (and **VerificationModule**)
- Attributes:
 - **PartOfProduct: Boolean**: This indicates that the document is part of the product. PMT checks that every document in the process is created and used. For artifacts that are part of product it suffices that they are created.
 - **ProjectInput: Boolean**: This indicates that the document is input to the process. PMT checks that every document in the process is created and used. For artifacts that are project input it suffices that they are used.
 - **Path: String**: Required to denote the path (absolute or relative) of the document, e.g. "QKit/TestSuite" or "C:\Qualification\Target\". The path is exported into VVT tool. It is used to group the checks and to determine if an element has changed and required re-verification/validation.
 - **Status: ProcessStatus**: Describes the status of the artifact. This can be used for project management, see Section 8.2.11.
 - **DocumentStatus: String**: Can be used to describe the status of the document, e.g. "Draft" or "Final". Currently this is not used in PMT.
 - **Format: String**: The format of the artifact, e.g. pdf.
 - **Version: String**: Can be used to describe the version of the document, e.g. "0.8" or "1.0". Currently this is not used in PMT.
- Associations:
 - **readByProcessModules: ProcessModule [0..*]**: The process modules that use the artifact as input.
 - **createdByProcessModules: ProcessModule [0..*]**: The process modules that create the artifact as output.
 - **DependsOnArtifacts: Artifact [0..*]**: Allows to model dependencies on the artifact level
 - **DependingArtifacts: Artifact [0..*]**: Allows to model dependencies on the artifact level
 - **createdByProcessModules: ProcessModule [0..*]**: The process modules that create the artifact as output.
 -
 - **StakeHolder: Stakeholder [0..1]**: The responsible stake holder for this artifact.
 - **InvolvedStakeHolders: Stakeholder [0..*]**: Other involved stakeholders (not the responsible one).

- **RefinedArtifact: Artifact [0..1]**: The artifact that is refined by this artifact, see Section 8.3
- **RefiningArtifacts: Artifact [0..*]**: The refining artifacts that specialize this artifact, see Section 8.3.
- **SubArtifactReferences: Artifact [0..*]**: the artifacts that are included via references here, see Section 8.2.5.
- **ReferencedByArtifacts: Artifact [0..*]**: The artifacts that include this artifact via reference, see Section 8.2.5.
- **Compliances: Compliance [0..*]**: the compliances that this artifacts contributes.
- Depreciated: **ProcessConditions: ProzessCondition [0..*]**: a condition where this artifact is involved.
- Compositions:
 - **SubArtifacts: Artifact [0..*]**: The artifacts that are contained in this artifact, e.g. the Test in the TestSuite or the chapters in the document.

9.10.3 StakeHolder -> Variantable

Stakeholders are the responsible (and contributing) roles / persons in processes. They “own” processes and artifacts. They can be modeled as depicted in Figure 172.

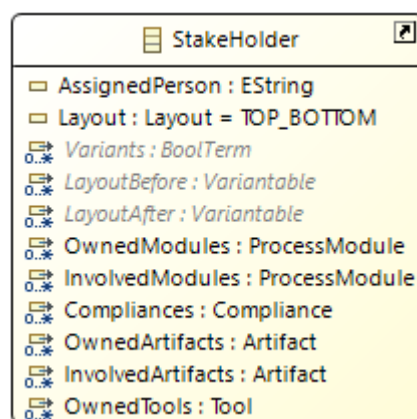


Figure 172: Meta-Model of StakeHolder with all Properties

StakeHolder has the following properties:

- Superclass:
 - **Variantable**, see Section 9.3.2.
- SubClasses: -
- Container: The following elements can contain StakeHolders:
 - **Process**
 - **ProcessModule** (and **VerificationModule**)
- Attributes:

- **AssignedPerson: String:** one person can be assigned to this. In case several persons shall be used, e.g. Testers, they can be listed in this field.
- Associations:
 - **ownedModules: ProcessModule [0..*]:** The process modules that the stakeholder is responsible for.
 - **InvolvedModules: ProcessModule [0..*]:** the process modules that this stakeholder is involved, but not responsible for.
 - **OwnedArtifacts: Artifact [0..*]:** the artifacts the stakeholder is responsible for.
 - **InvolvedArtifacts: Artifact [0..*]:** the artifacts that this stakeholder is involved, but not responsible for.
 - **Compliances: Compliance [0..*]:** the compliances that this stakeholder contributes.
 - **OwnedTools: Tool [0..*]:** the tools that this stakeholder is responsible for.
- Compositions: -

9.10.4 VerificationModule -> ProcessModule

Verification Modules are a special case of ProcessModules. They have been introduced since we decided that every requirement has to be verified and there needs to be a formal difference between a generic process and a verification module. A verification module has Criteria (checks/questions) that have to be verified within the project. The verification modules are the elements that are the inputs for the verification and validation tool (VVT). VerificationModules verify always something, usually an artifact.

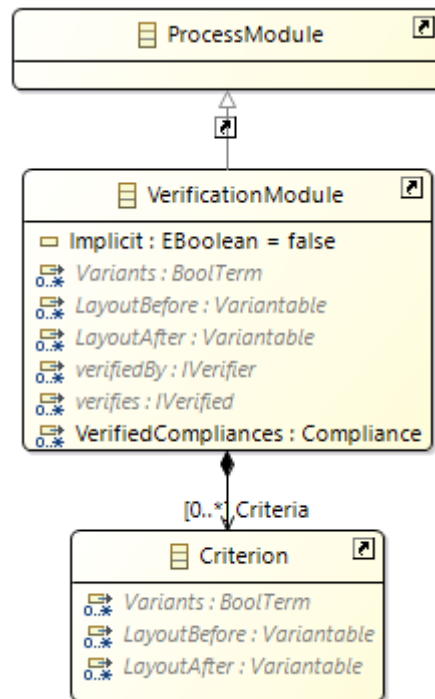


Figure 173: Meta-Model of VerificationModule with all Properties

In addition to ProcessModule the VerificationModule has the following properties:

- Superclass:
 - **ProcessModule**, see Section 9.10.1.
- SubClasses: -
- Container: The following elements can contain VerificationModules:
 - **Process**
 - **ProcessModule** (and **VerificationModule**)
- Attributes:
 - **Implicit: Boolean**: indicates that the verification is done implicitly, i.e. without further action required. In this case this verification module should not have criteria, but Sub-Modules (modeled as Sub-Processes).
- Associations:
 - **VerifiedCompliances: Compliance [0..*]**: the compliances that this verification module explicitly verifies. Note that this is the “Mandatory” relation to Compliance, because every Compliance element has to have ProcessModules that implement them (usually not verification modules) and VerificationModule (or Criterion) to verify them.
- Compositions:
 - **Criteria: Criterion [0..*]**: The criteria to be checked when performing this verification and validation.

9.10.5 Criterion -> Variantable

A Criterion is a single question that has to be answered as part of a verification, for example: "Is the specification clear?" or "Has the test been executed successfully?"

Criterion has the following properties:

- Superclass:
 - **Variantable**, see Section 9.3.2.
- SubClasses: -
- Container: The following elements can contain VerificationModules:
 - **VerificationModule**
- Attributes: -
- Associations:
 - **VerifiedCompliances: Compliance [0..*]**: the compliances that this criterion explicitly verifies. Note that this is the "Mandatory" relation to Compliance, because every Compliance element has to have ProcessModules that implement them (usually not verification modules) and VerificationModule or Criteria to verify them.
- Compositions: -

Note: that the "question" of the criterion has to be modeled within its description. So a good example of a Criterion is:

- Name: "Clearness"
- ID: "C1"
- Description: "Is the specification clear?"

9.11 Requirements & Compliance

Requirements and compliance are core functionalities of PMT. Their handling is described in Section 8.2.3 and 8.2.4. The meta model for Requirements & Compliances is depicted in Figure 174.

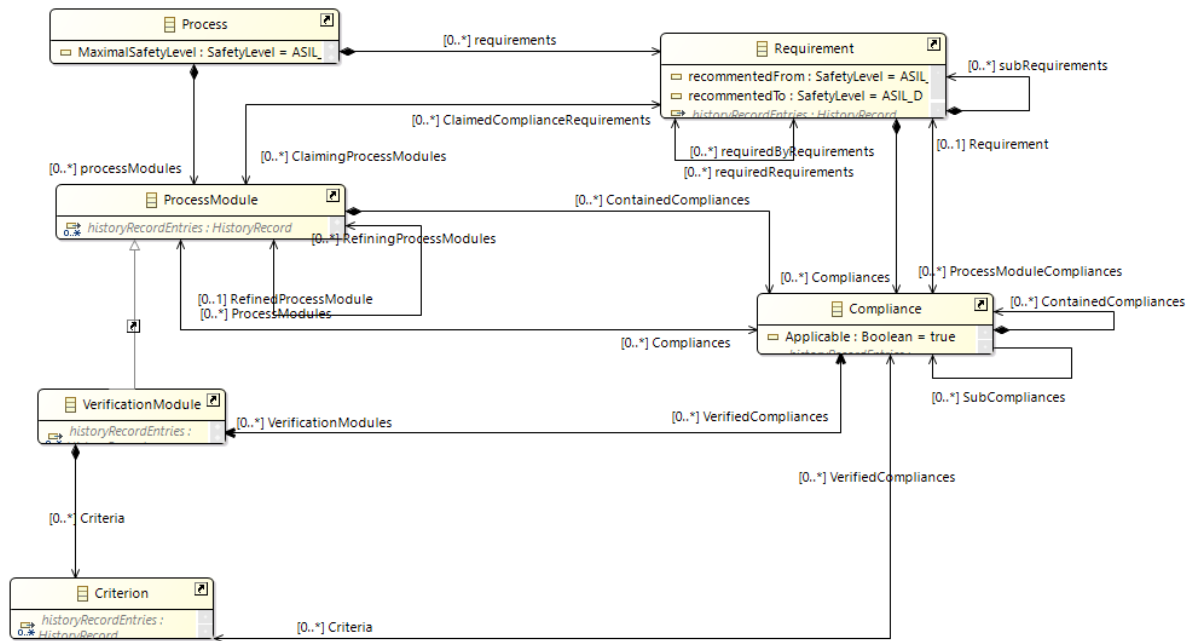


Figure 174: Metamodel for Requirements & Compliances

9.11.1 Requirement -> Variantable

The requirement describes requirements for the processes, usually derived / copied from safety standards. Important is to keep the traceability back to the standards. This is typically done by using the IDs from the standards as IDs.

Requirement has the following properties:

- Superclass:
 - **Variantable**, see Section 9.3.2.
- SubClasses: -
- Container: The following elements can contain Requirements:
 - **Process**: Top level requirements like "ISO 26262" are usually contained in the process element
 - **Requirement**: Sub-Requirements, like "8-11" are usually contained in other requirements.
- Attributes:
 - RecommendedFrom: SafetyLevel: the lower level where this requirement is mandatory from. This corresponds to "highly recommended" in most standards.
 - RecommendedFrom: SafetyLevel: the upper level where this requirement is mandatory.
- Associations:
 - **RequiredRequirements: Requirement [0..*]**: The required requirements. Usually all contained requirements are by default

also “required”. This association can be used to implement references to other chapters

- **RequiredByRequirements: Requirement [0..*]:** the inverse relation of required requirements (see above).
 - **ClaimingProcessModules: ProcessModule [0..*]:** the process modules that claim to satisfy this requirement.
 - **ProcessModuleCompliances: Compliance [0..*]:** the compliance argumentations for this requirements that are contained in ProcessModules.
- Compositions:
 - **SubRequirements: Requirement [0..*]:** The contained requirements usually the sub-sections of a section or the requirements in a section.
 - **Compliances: Compliance [0..*]:** The compliance argumentation explaining how this requirements is satisfied and verified.

9.11.2 Compliance -> Variantable

The compliance element is used to express the compliance with a requirement (either the containing or a linked requirement). Compliance consists of a reasonable argumentation (in the description) and three related things:

- Requirement: the requirement that is satisfied
- ProcessModules: the ProcessModules that implement the requirement (also Stakeholders or Artifacts can be used to implement the requirement).
- VerificationModule: the verification module that checks (using criteria) if the requirement is satisfied.

Compliance has the following properties:

- Superclass:
 - **Variantable**, see Section 9.3.2.
- SubClasses: -
- Container: The following elements can contain Requirements:
 - **ProcessModule:** ProcessModule can contain compliance argumentations in a modular/reusable way.
 - **Requirement:** Each requirements can contain it's compliance argumentation.
- Attributes:

- Applicable: Boolean (default=true): indicates if the Requirement is applicable¹².
- Associations:
 - **ProcessModules: ProcessModule [0..*]:** the process modules that implement this requirement.
 - **StakeHolders: StakeHolder [0..*]:** the stake holders that implement this requirement.
 - **Artifacts: Artifact [0..*]:** the artifacts that implement this requirement.
 - **RequiredRequirements: Requirement [0..*]:** The required requirements. Usually all contained requirements are by default also “required”. This association can be used to implement references to other chapters
 - **Requirement: Requirement [0..1]:** the requirement that this compliance is arguing (Note this is only one, since every requirements should have it’s own argumentation). In case the compliance is contained in a Requirement, this association is not needed.
 - **SubCompliances: Compliance [0..*]:** the compliance that are logically required to argue the compliance of the requirements.
 - **VerificationModules: VerificationModule [0..*]:** the verification modules that verify that this Compliance argumentation is true.
 - **Criteria: Criterion [0..*]:** the Criteria that verify that this Compliance argumentation is true.

Note every Compliance Argumentation has to have either Sub-Compliances or VerificationModules or Criteria.

- Compositions:
 - **ContainedCompliances: Compliance [0..*]:** the compliance that are contained in this compliance, per default contributing to the compliance.

¹² Usually we try to avoid non-applicable requirements, e.g. by tailoring using variants, however sometimes it is easier to mark a requirement as no applicable using this attribute.

9.12 Model-Based Processes

Model-Based processes usually come with a modeling tool. This modeling tool has a meta model that formalizes the models that can be built syntactically.

The elements to describe model-based processes are (see Figure 175):

- Model: Specialization of Artifact containing the modeling elements
- MetaModel: Container for the meta-model (in order to make it reusable in different models)
- MetaModelElement: the modeling elements that can be used to create the model
- MetaModelAttribute: the attributes of the elements that can be used to describe the elements
- MetaModelAssociation: the associations of the elements that can be used to describe the relations between elements

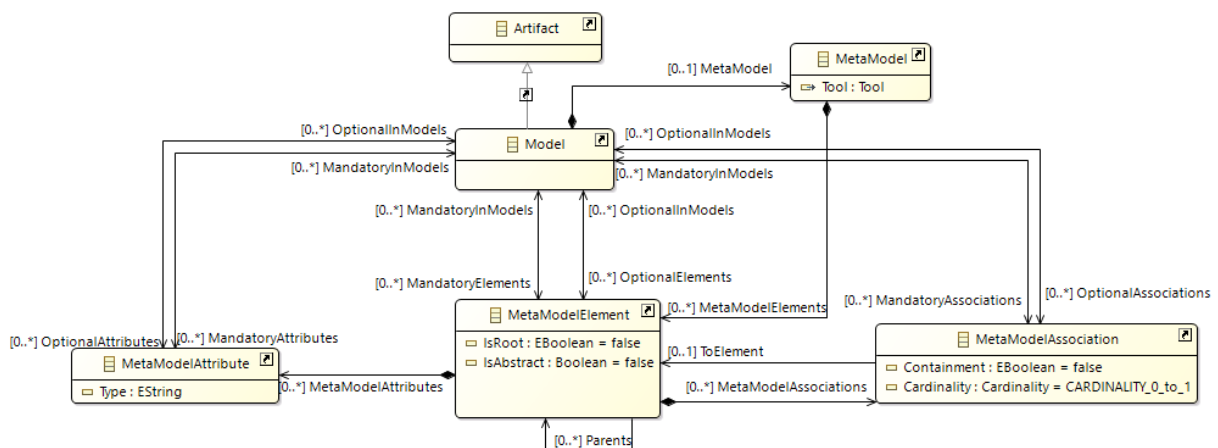


Figure 175: Meta-Model for Model-Based Processes

In contrast to other UML models & modeling tools, e.g. EMF, the goal of this model is not to generate code from the meta model, but just to describe the model. Therefore some aspects do not need to be modeled, e.g. the inverse relations or the cardinalities¹³ of the associations. Those things are part of the modeling tools that are described.

9.12.1 Model -> Artifact

The model element is a description of the models that are created within a model-based process. In contrast to an Artifact it allows to provide a detailed specification of a model element including a specification of mandatory and optional elements.

Model has the following properties:

- Superclass:

¹³ Cardinalities have been added to the model for description purpose only.

- **Artifact**, see Section 9.3.2.
- SubClasses: -
- Container: The following elements can contain Models:
 - **Process**
 - **ProcessModule** (and **VerificationModule**)
- Attributes: -
- Associations:
 - **MandatoryElements: MetaModelElement [0..*]**: The mandatory elements that have to be used when creating the model
 - **OptionalElements: MetaModelElement [0..*]**: The optional elements that can be used when creating the model.
 - **MandatoryAttributes: MetaModelAttribute [0..*]**: The mandatory attributes that have to be used when creating the model.
 - **OptionalAttributes: MetaModelAttribute [0..*]**: The optional attributes that can be used when creating the model.
 - **MandatoryAssociations: MetaModelAssociation [0..*]**: The mandatory associations that have to be used when creating the model.
 - **OptionalAssociations: MetaModelAssociation [0..*]**: The optional associations that can be used when creating the model.
- Compositions:
 - **MetaModel: MetaModel [0..1]**: the container for the meta model

Note: Models have to be created manually within PMT, however the meta model can be imported automatically for modeling tools built with Eclipse, since it is possible to import .ecore files and to create the corresponding models, see ("Ecore Import") in Section 7.2.4.

9.12.2 MetaModel

The meta model is the meta model of a used modeling tool. It can be used to build different kinds of models, for example an UML Tool can have state charts and class diagrams that both use packages. Therefore the meta model is a separate entity that can be reused within all models of that tool.

MetaModel has the following properties:

- Superclass:
- SubClasses: -
- Container: The following elements can contain MetaModel elements:
 - **Model**

- Attributes: -
- Associations: -
 - **Tool: Tool [0..1]:** The modeling tool that is based on this meta model.
- Compositions:
 - **MetaModelElements: MetaModelElement [0..*]:** The elements in the meta model.

9.12.3 MetaModelElement -> Named

The meta model element corresponds to a single element in the modeling tool, e.g. a class or a state.

MetaModelElement has the following properties:

- Superclass:
 - **Named**, see Section 9.3.1.
- SubClasses: -
- Container: The following elements can contain MetaModelElements:
 - **MetaModel**
- Attributes: -
 - **IsRoot: Boolean:** Characterizes the root element in the model that is the root of the model tree. In PMT "Process" is the root element for all models.
 - **IsAbstract: Boolean:** Is true for interfaces or abstract classes.
- Associations:
 - **MandatoryInModels: Model [0..*]:** The models in which this element is mandatory.
 - **OptionalInModels: Model [0..*]:** The models in which this element is optional.
 - **Parents: MetaModelElements [0..*]:** The parent/superclass elements/interfaces of this model element. For Example "MetaModelElement" has the parent "Named".
- Compositions:
 - **MetaModelAttributes: MetaModelAttribute [0..*]:** The attributes of the model element.
 - **MetaModelassociations: MetaModelAssociation [0..*]:** The associations/relations of the model element.

9.12.4 MetaModelAttribute -> Named

Attributes are used to specify properties of elements. Using the element MetaModelAttribute this can be specified. MetaModelAttribute has the following properties:

- Superclass:
 - **Named**, see Section 9.3.1.

- SubClasses: -
- Container: The following elements can contain MetaModelAttributes:
 - **MetaModelElement**
- Attributes: -
 - **Type: String:** Describes the type of the attribute, e.g. String or Boolean.
- Associations:
 - **MandatoryInModels: Model [0.*]:** The models in which this attribute is mandatory.
 - **OptionalInModels: Model [0.*]:** The models in which this attribute is optional.
- Compositions: -

9.12.5 MetaModelAssociation -> Named

Associations are used to describe the relations between elements. They can be compositions and non-containing associations. Sometimes associations denote only non-compositional relations. MetaModelAssociation has the following properties:

- Superclass:
 - **Named**, see Section 9.3.1.
- SubClasses: -
- Container: The following elements can contain MetaModelAssociations:
 - **MetaModelElement**
- Attributes:
 - **Containment: Boolean:** Is true for associations that describe compositions, i.e. contain other elements in the modeling tree.
 - **Cardinality: Cardinality:** Specifies the cardinality of the associations (currently this is only used for description purpose).
- Associations: -
 - **MandatoryInModels: Model [0.*]:** The models in which this association is mandatory.
 - **OptionalInModels: Model [0.*]:** The models in which this association is optional.
- Compositions: -

9.13 Tools

Tools play an important role in processes and support methods¹⁴. They can be modeled as depicted in Figure 176.

¹⁴ Note that the use of some methods is required from safety standards, however for simplicity we decided not to make a so formal model of the standards that would include methods and their safety requirements. This can be done in the TCA tool when refining the model. In PMT Methods have currently no special semantics and can be modeled only for User Manual of Process Modeling Tool
Version 1.2

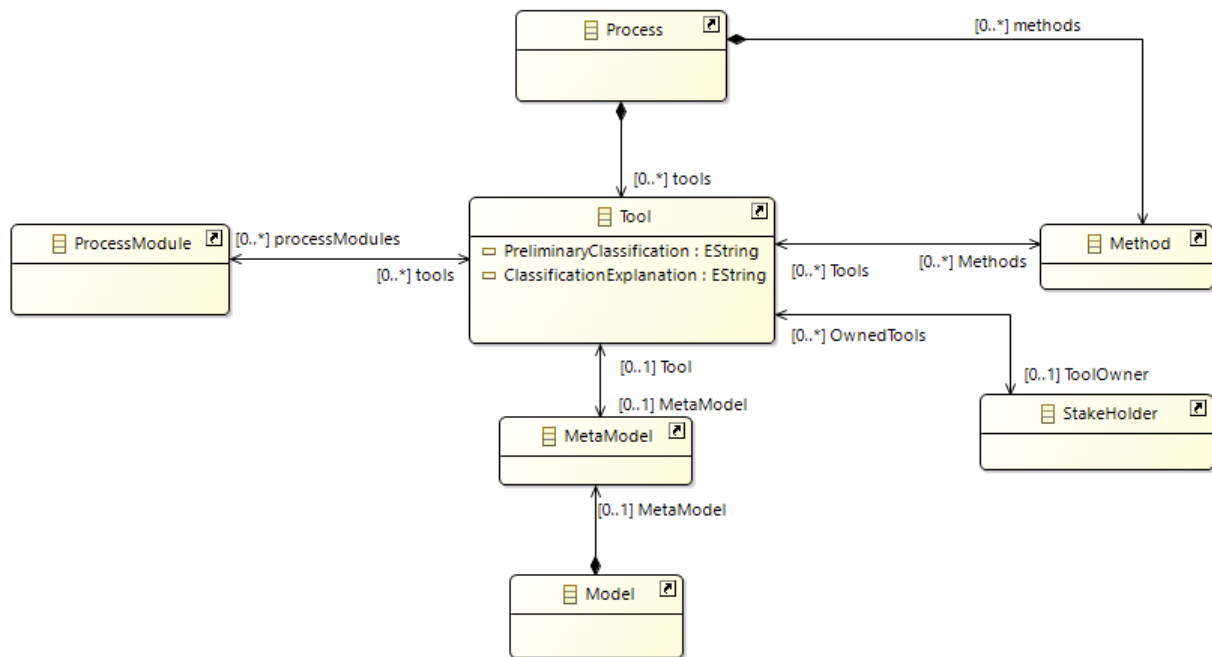


Figure 176: Meta-Model of Tools

9.13.1 Tool -> Variantable

Tools can be used to support the application of methods in processes. In safety relevant processes it is important to use the tools safely. Therefore they have to be classified and eventually qualified. Tool has the following properties:

- Superclass:
 - **Named**, see Section 9.3.2.
- SubClasses: -
- Containers:
 - **Process:** Tools can only be contained in the global container.
- Attributes:
 - **Preliminary Classification: String:** a preliminary classification of the tool, for example "TCL1", "uncritical" or "To be qualified".
 - **Classification Explanation: String:** Should explain the preliminary¹⁵ classification.
- Associations: -
 - **ProcessModules: ProcessModule [0..*]:** The process modules ("use cases") where the tool is used.
 - **Methods: Method [0..*]:** the supported methods from the tool.

documentation purpose.

¹⁵ Note this is not a standard compliant way to classify tools, but might serve as a first orientation. Use the TCA tool from <http://www.validas.de/en/services/tca/> for standard compliant classification and qualification. Future versions of TCA might be able to import PMT models for refinement.

- **ToolOwner: StakeHolder [0..1]**: The owner responsible for the tool (and its qualification).
- Compositions: -

9.13.2 Method -> Named

Methods can be required from safety standards and applied within processes. Method has the following properties:

- Superclass:
 - **Named**, see Section 9.3.1.
- SubClasses: -
- Containers:
 - **Process**: Methods can only be contained in the global container
- Attributes: -
- Associations: -
 - **Tools: Tool [0.*]**: The tools that support this method.
- Compositions: -

10 Known Issues

The known issues can be accessed at the bug tracing system of the <https://validas.atlassian.net/projects/PMT/issues/> (internally). Issues can be submitted by sending <mailto:info@validas.de>

The errors are classified with a priority (blocker=highest, critical=high, major=medium, minor=log, and trivial=lowest).

For the version 1.0 the following known bugs with priority major or higher are known (at April 17th, 2019). The corresponding work arounds are described if applicable.

- PMT-16: Preferences
- PMT-39: EMF Form of Process does not collapse
- PMT-38: [Cannot Delete Types and Parameters](#)
- PMT-20 Scroll-Bars in Forms not OK
- PMT-9 StackOverflow when opening inconsistent Model

None of the currently there are no blockers available, therefore this version of PMT was released.

Please report issues found during the work with the PMT to info@validas.de.

11 Licenses & Liability

The PMT/VVT are products of Validas AG and must not be distributed without permission of Validas AG.

It has been developed using Eclipse and POI and docx4j. The licenses of these components are:

- Eclipse: EPL: <http://www.eclipse.org/legal/epl-v10.html>
- POI: Apache License Version 2.0: <http://www.apache.org/licenses/>
- docx4j: Apache License Version 2.0

16

Validas AG does not take any guarantee for the functionality of the PMT / VVT tool. As stated in the previous section, PMT has a TCL 1 and might have critical errors that the user has to detect during the review of the results.

VALIDAS AG AND ITS AFFILIATES MAKE NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. VALIDAS AG AND ITS AFFILIATES SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF VALIDAS AG AND ITS AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

¹⁶ Note the Apache Licence 2.0 is distributed with this product in the jar file of the plugin de.validas.excelinterface, which is found in the plugins directory of the PMT.

12 Examples & Further Documentation

In the distribution of the tool chain analyzer is an examples directory. It can be found at `<PMT>/plugins/de.validas.spm.pmt.examples_1.0.0<date>/`.

It contains the following Models:

- ModuleTest: Model and artifacts for a module test process and it's compliance to ISO 26262
- MetaProcess: Process Model for the creation of the process
- Documentation/*: models for images and examples used within the user manual

In addition to this user manual, there is a First Step presentation available "PMTEExample.ppt" in the example plugin. It is located in

- ModuleTest/PMTEExample.ppt

Furthermore there are (currently German only) video tutorials available for download explaining the method. They can be found in the PMT section of <http://www.validas.de/en/services/qualification/>.