



# Trusted Computing Engineering for Resource Constrained Embedded Systems Applications

**Deliverable 4.4**

**Repository V2**

**Project:** TERESA  
**Project Number:** IST-248410  
**Deliverable:** D4.4  
**Title:** Repository V2  
**Version:** v1.6  
**Confidentiality:** Public  
**Editors:** B. Hamid (IRIT), A. Ziani (IRIT)  
**Date:** 20.01.2013



Part of the Seventh  
Framework Program  
Funded by the EC - DG INFSO

## Control Sheet

<b>Approval</b>		
	Name	Date
Prepared	Brahim Hamid and Adel Ziani	10.01.2013
Reviewed	All Project Partners	20.01.2013
Authorized	Antonio Kung	20.01.2013
<b>Circulation</b>		
Recipient	Date of submission	
Project Partners	20.01.2013	
European Commission	20.01.2013	

<b>Version History</b>			
Version number	Date	Main author	Summary of changes
v1.0	01.11.2012	Brahim Hamid and Adel Ziani -IRIT	Initial version
v1.1	10.11.2012	Brahim Hamid and Adel Ziani -IRIT	Metamodels revisions
v1.2	10.12.2012	Brahim Hamid -IRIT	DSLs specifications
v1.3	10.12.2012	Brahim Hamid -IRIT	Repository specifications
v1.4	20.12.2012	Adel Ziani -IRIT	DSLs editors implementations
v1.5	10.01.2013	Adel Ziani -IRIT	Repository implementation
v1.6	20.01.2013	All	Internal Review

# Contents

<b>Executive Summary</b>	<b>7</b>
<b>1 Introduction</b>	<b>8</b>
<b>2 Feedback Analysis</b>	<b>9</b>
2.1 Feedback Template . . . . .	9
2.2 TERESA BugTrack System . . . . .	10
2.3 Requests for the Evolution of the Languages and the Tools . . . . .	10
<b>3 Repository System Implementation</b>	<b>12</b>
3.1 Implementation of the Repository . . . . .	12
3.1.1 CDO Repository Implementation . . . . .	13
3.2 Repository Interfaces . . . . .	16
3.2.1 Common API . . . . .	17
3.2.2 API for Administration . . . . .	17
3.2.3 API for Languages . . . . .	18
3.2.4 API for Property . . . . .	18
3.2.5 API for Pattern . . . . .	19
3.3 Gaya Product . . . . .	19
3.3.1 Installation . . . . .	19
3.3.2 Gaya License . . . . .	20
<b>4 Repository Management</b>	<b>21</b>
4.1 User Management . . . . .	22
4.2 Compartment Management . . . . .	23
4.3 Authentication . . . . .	24
4.4 Property Management . . . . .	25
4.5 Pattern Management . . . . .	25
<b>5 Repository Populating</b>	<b>26</b>
5.1 Property Designer . . . . .	26
5.1.1 Property Validation . . . . .	28
5.1.2 Property Deposit . . . . .	28
5.1.3 Tiqueo Product . . . . .	30
5.1.4 Tiqueo License . . . . .	31
5.2 Pattern Designer . . . . .	32
5.2.1 Modeling Pattern at DIPM . . . . .	32
5.2.2 Modeling Pattern at DSPM . . . . .	34

---

5.2.3	Pattern Conformance Validation . . . . .	36
5.2.4	Generation of Documentation . . . . .	36
5.2.5	Pattern Deposit . . . . .	37
5.2.6	Validation Artifacts . . . . .	39
5.2.7	Arabion Product . . . . .	39
5.2.8	Arabion License . . . . .	41
5.3	Repository Retrieval . . . . .	42
5.3.1	Property Retrieval . . . . .	42
5.3.2	Pattern Retrieval . . . . .	43
5.4	An Overview of the TERESA Repository Content . . . . .	45
<b>6</b>	<b>Outlook</b>	<b>48</b>
6.1	Web Service Implementation of the Gaya Repository . . . . .	48
6.2	Graphical Pattern Editor . . . . .	49
<b>7</b>	<b>Conclusion</b>	<b>51</b>
	<b>Appendix A: Terminology and Abbreviation</b>	<b>51</b>
	<b>Appendix B: Tiqueo Examples</b>	<b>53</b>
	<b>Appendix C: Arabion Examples</b>	<b>54</b>

## List of Figures

2.1	The Bug Track view - Gaya	10
3.1	Integrated Modeling/Configuration/Assembly Process	13
3.2	An Overview of the Tools Components	13
3.3	Overview of the Tool Suite Implementation - Example of Gaya Repository and APIs	14
3.4	Gaya CDO based architecture	15
3.5	The Repository Interfaces and Classes	16
3.6	API for Pattern	17
4.1	The Admin UI of the Repository	21
4.2	User Management Part	22
4.3	Compartment Management	23
4.4	Repository Authentication	24
4.5	Repository Authentication Under Eclipse	24
4.6	Property management part	25
4.7	Pattern management part	25
5.1	P&C Library Development	27
5.2	Type and Category Libraries Definition Processes	27
5.3	Designing a Category Library	28
5.4	Eclipse Load Resource Tool	29
5.5	Property Validation	29
5.6	Property Deposit	30
5.7	Pattern development process at DIPM	33
5.8	Secure Communication DI Pattern at Design level	34
5.9	Secure Communication DS pattern at Design level	35
5.10	Eclipse Load Resource Tool	36
5.11	Pattern Validation	37
5.12	Generation of Documentation	37
5.13	HTML Documentation	38
5.14	Pattern Publication	38
5.15	Pattern Validation Artifact -a	39
5.16	Pattern Validation Artifact -b	40
5.17	Property Library Instantiation	42
5.18	Pattern Instantiation	44
5.19	Pattern Instantiation - Consistency	44
5.20	Repository Content	45

---

6.1	Gaya Webservice based architecture . . . . .	49
6.2	Pattern Designer with a Graphical Environment . . . . .	50
7.1	Railway S&D Unit Library . . . . .	54
7.2	Railway S&D Type Library . . . . .	54
7.3	Railway S&D Property Category Library . . . . .	55
7.4	Railway Resource Unit Library . . . . .	56
7.5	Railway Resource Type Library . . . . .	57
7.6	Railway Resource Property Category Library . . . . .	57
7.7	DIPattern TMR System . . . . .	58
7.8	DIPattern TMR Architecture . . . . .	59
7.9	DSPattern TMR System . . . . .	60
7.10	DSPattern TMR Architecture . . . . .	61

## Executive Summary

D4.4 is a document presenting the new version of the TERESA model-based repository implementation. The structure, the interactions and the content are improved to meet the TERESA needs. After the discussion on the lacks of the previous version, we describe the new implementation of the repository and its related APIs using Eclipse CDO Technology. Then, we detail the back office part of the repository including the set of tools for the management of the repository. The next parts present the tool suite for populating the repository with patterns and property model libraries. Further, we discuss the generation of documentation, the validation artefacts and some ongoing experimentations.

A partial results of Section 5.2 is published as:

- Towards a Security and Dependability Pattern Development Technique for Resource Constrained Embedded Systems, in the Software Quality, Process Automation in Software Development (SWQD 2012), with N. Desnos, B. Hamid, C. Percebois and D. Gouteux.

Some of the results in this deliverable that are not yet been published are:

- A partial results of Section 3.1 is submitted to the International Conference on Software Reuse as: A Model-based Repository of S&D Patterns for RCES.
- The results of this deliverable related to the structure and the implementation of the repository of modeling artifacts is submitted to the Springer STTT journal as: Model-Driven Engineering Trusted Embedded Systems based on a Repository of S&D Patterns.

This deliverable improves the repository implementation presented in the D4.3, offering the new following features:

- New storage organization: More efficient use of the Eclipse CDO Technology(XMI resource referencing,...),
- Access control management,
- Administration: UI for adding new users and compartments for new domains,
- Logging management,
- Extending the APIs to support the exchange of pattern attached documents, pattern validation artefacts, ... ,
- Retrieval tools: one for each type of the repository modeling artefacts,
- Documentation,
- Code documentation and User manual

# 1 Introduction

---

The main goal of the *WP4* is to define a modeling and development framework to support the specification, the definition and the packaging of a set of modeling artifacts to assist the developers of trusted applications for Resource Constrained Embedded Systems (RCES). In the context of the TERESA project, we have identified three kinds of modeling artifacts. Here, we deal with Security&Dependability (S&D) and resource properties.

In this deliverable we describe the novel implementation of the TERESA repository conforming to the specification languages described in the Deliverable D4.2. It constitutes an evolution to the implementation presented in the Deliverable D4.3 regarding TERESA's partner requests. Most of the expectations are met. We discussed how the TERESA model-based repository is integrated in the Model Driven Engineering (MDE) approach, mainly from the tool-chain perspective. This version is also based on the Eclipse technology, mainly on Eclipse Modeling Framework Technologies (EMFT<sup>1</sup>) as open source DSML environment.

The rest of this document is organized as follows. In Chapter 2 we discuss the feedback about the first version of the model-based repository's specification and implementation. In Chapter 3, we present the new implementation of the repository following the new structure presented in the reviewed version of the Deliverable D4.2. Chapter 4 details the repository management while Chapter 5 presents the tool suite for the assistance of modeling artifact design for populating the repository and the documentation generation. Then, in Chapter 6, we highlight some experimentations for the tool suite evolution for external dissemination and exploitation. Finally, Chapter 7 concludes the deliverable with the reminder of the achieved work. Appendices are added with a list of abbreviations and acronyms, as well as some examples.

---

<sup>1</sup><http://www.eclipse.org/modeling/emft/>



## 2 Feedback Analysis

In this chapter, we discuss the feedback from the TERESA's industrial partners to improve the tool suite (based on the Eclipse platform), including the editors to create the TERESA modeling artifacts and the repository for reuse, presented in the Deliverable D4.3. To this end, we provided in the Deliverable 4.3 a template as a set of measures to evaluate the TERESA set of languages and tools. The results are used as a first step for the assessment of the TERESA approach. In addition, the tool suite was accompanied with a bugtracking system to test the TERESA tool suite features and to give feedback.

### 2.1 Feedback Template

The TERESA tool suite was proposed for evaluation to the TERESA consortium. The tool suite is provided as Eclipse plugins by an Eclipse conform *p2-repository* under: <http://www.semcomdt.org/semco/tools/updates/1.2>. The following describe the evaluation.

In the Deliverable D4.3, we proposed a template to measure the acceptance of the modeling languages and tools we offer. We have identified a set of measures to evaluate the usage of the models and the user-friendliness of the tools. The evaluation template is based on TAM (Technology Acceptance Model) and concerns the specification languages (Property, Patterns and the Repository) as well as the tools (Tiqueo, Arabion and Gaya). We asked participants to give scores from 1 to 5 (5 is the best). We first evaluated the perceived usefulness of the solution itself (items 1-4). Next, we focus on the ease of the solution (items 5-6). Finally, we want also to measure the compatibility of the solution with existing environments. (items 7-9). These scores indicates the degree of satisfaction of the users and provides a feedback to us in order to enhance our specification languages and the tool suite. The following table depicts an overview of the results of our experiment.

Item	Mean	St. Deviation
1. Design quality	3.5	0.4
2. Model completeness	4	0.3
3. Documentation and code generation readability	4	0.89
4. Effort spent on development	4.5	0.16
5. Model understandability	3.5	0.475
6. Effectiveness	3.5	0.6
7. Integration with other solutions	4	0.86
8. Standards compliance	4	0.2
9. Cost of adoption	3.5	0.48

## 2.2 TERESA BugTrack System

For managing our development process, we set up a bugtracking system using the *buggenie*<sup>1</sup> framework. The system is submitted to the TERESA consortium under: <http://www.semcomdt.org/bugtracker/>, providing a set of topics: arabion, tiqueo, naravas, gaya . . . . Each topic consists of a cluster of information to describe the status of a one tool development process in more detail.

As visualized in Figure 2.1, the view of the Gaya development status (as of January 2013) provides the following information.

Among the 15 submitted requests:

- 9 requests were fixed,
- 2 requests were not a bug,
- we are working on 2,
- we are investigating 1,
- we received 1 new request.

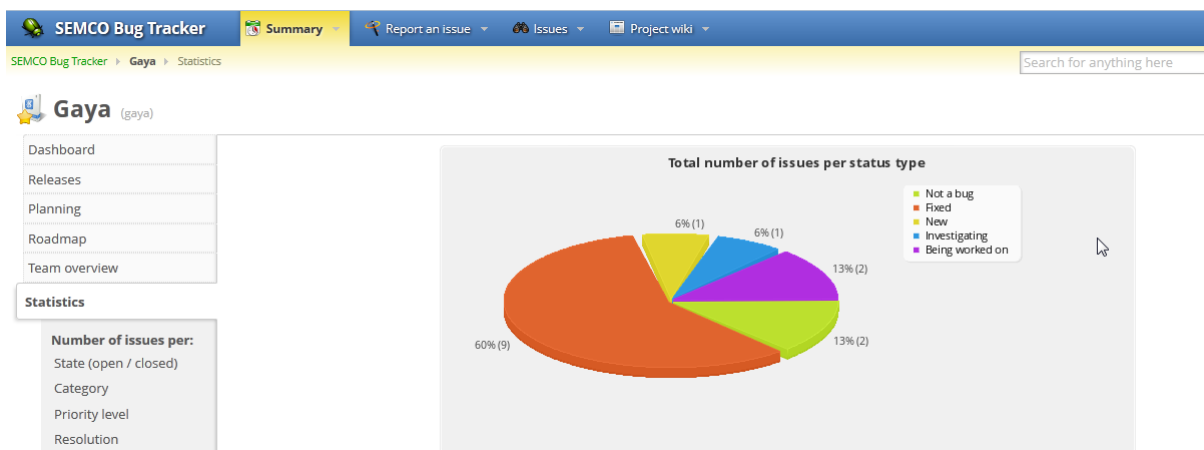


Figure 2.1: The Bug Track view - Gaya

## 2.3 Requests for the Evolution of the Languages and the Tools

In the following we summarize TERESA's partners' principal requests related to the languages and the tools presented in the Deliverable D4.3.

<sup>1</sup><http://www.thebuggenie.com>

- *Property.*
  - Support complex types,
  - Add more primitive types,
  - Support types as list of choices,
  - Add icons to the Tiqueo EMF editor for each of the property concepts to make it more intuitive.
- *Pattern.*
  - Internal structure should be specified by external tools (e.g, UML editors),
  - Document artifact should allow adding information not captured by the existing concepts (e.g, libraries),
  - Validation artifacts should be linked to the pattern as external document,
  - Add icons to the Arabion EMF editor for each of the pattern concepts to make it more intuitive.
  - Propose a Graphical version of Arabion
- *Repository.*
  - Administration (user, compartment and logging management): add new users, access control management, add storage compartments for new domains,
  - Extend the features of the GAYA APIs
    - \* API4Pattern: Extend the features of the SearchPattern function of the PatternAPI to take into account the following parameters, i.e adapt the parameters of the query: phase, S&D category, Resource category, Keywords, DI, DS, name, alsoKnowsAs, validated patterns, ...
    - \* API for validation artifacts: adapt the API to support the validation artifact during the publication and the download of a pattern

## 3 Repository System Implementation

Using the proposed metamodels, described in Deliverable 4.2, and Eclipse Modeling Framework (EMF) <sup>1</sup>, ongoing experimental work is done with *semcomdt* <sup>2</sup> (SEMCO Model Development Tools, IRIT's editor and platform plugins) testing the features:

- *Gaya G.* for the repository structure and interfaces conforming to *SARM*,
- *Gaya AdminGadmin.* for the repository administration,
- *Tiqueo (T).* for specifying models of S&D properties conforming to *GPRM*,
- *Arabion (A).* for specifying patterns conforming to *SEPM*,
- *Retrieval (RET).* for the repository access.

For instance, Fig. 3.1 shows the environment of the integrated modeling, storage and system development process based on our vision and approach.

Figure 3.2 presents our implementation architecture. In this Chapter, we focus on the *Gaya* repository and its interfaces. In the next two chapters, we present the repository management tools and the repository populating tools respectively.

### 3.1 Implementation of the Repository

In Figure 3.3, we illustrate the usage of a DSL building process, as introduced in the Deliverable 4.3, based on MDE technologies to define the DSLs for the modeling artifacts specification languages, followed by the one dedicated to build the *Gaya* repository and its APIs. Then, we provide the environment for the use of *Gaya* repository to store the modeling artifact specifications and instances through the APIs. The APIs are provided to end users, for instance artifact designers and system developers. The implementation is derived from the repository model and implemented using Java and the Eclipse CDO Server Technology.

---

<sup>1</sup><http://www.eclipse.org/modeling/emf/>

<sup>2</sup><http://www.semcomdt.org>

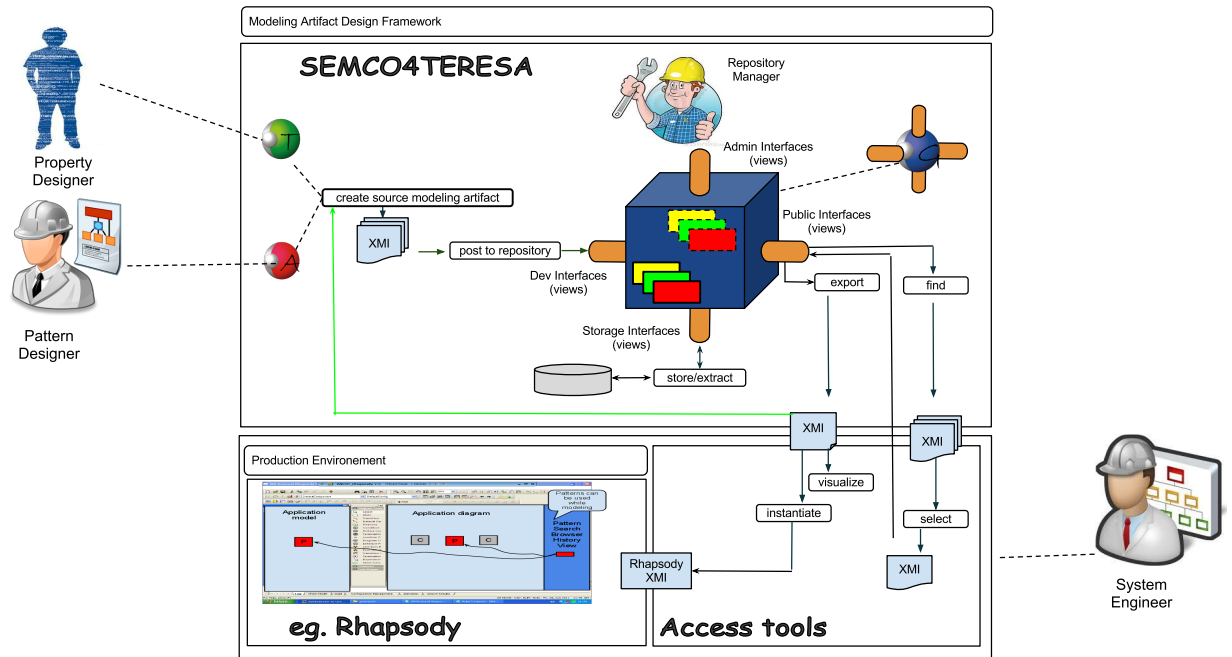


Figure 3.1: Integrated Modeling/Configuration/Assembly Process

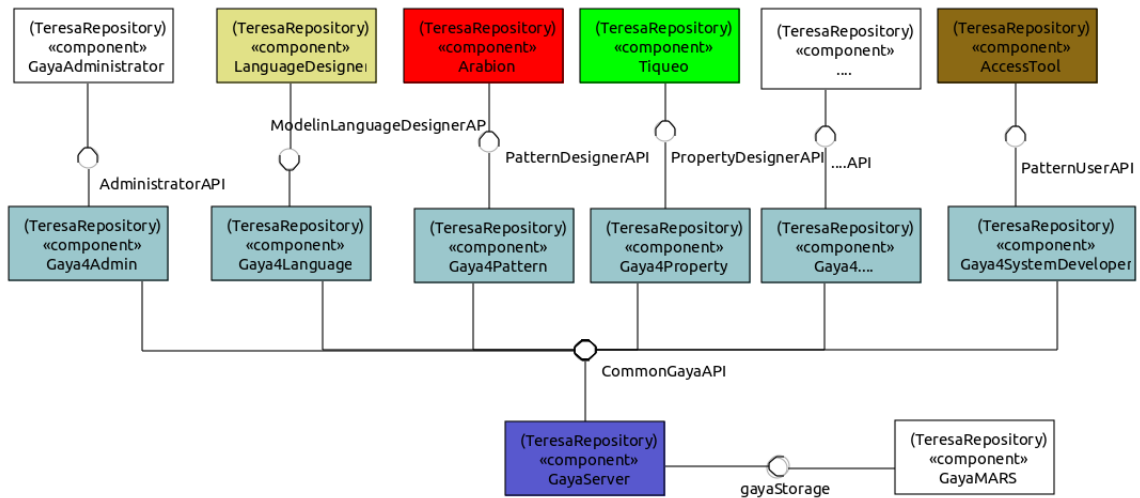


Figure 3.2: An Overview of the Tools Components

### 3.1.1 CDO Repository Implementation

We used the Eclipse EMF/CDO based Ecore technology to create our repository system as shown in Figure 3.4. The light gray blocks are the tools that constitute the architecture of the CDO repository and ensure the entire functioning of the server and clients. The

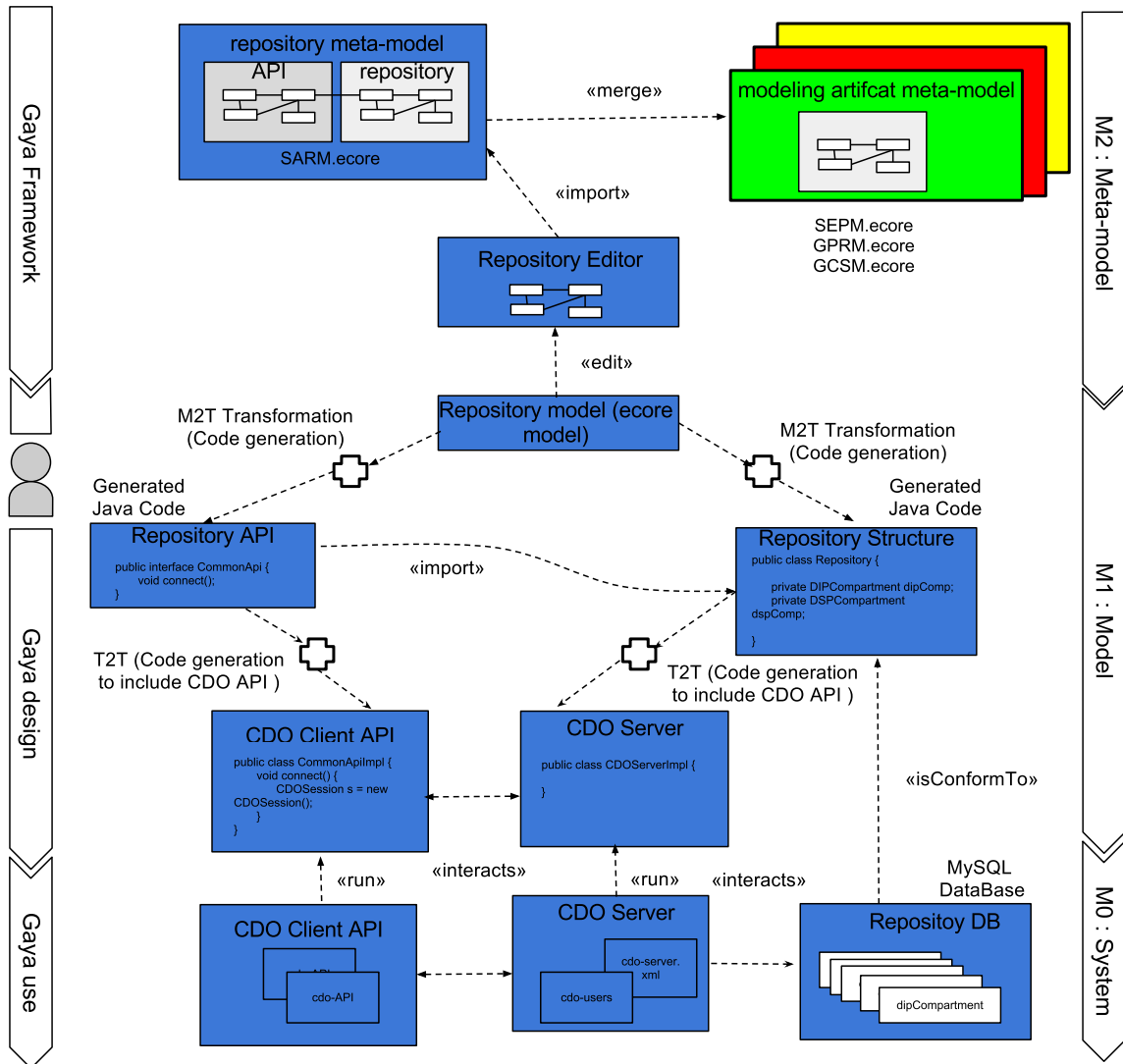


Figure 3.3: Overview of the Tool Suite Implementation - Example of Gaya Repository and APIs

dark gray block represents the models to be stored in the repository. The brown colored block represents the API defined to interact with the repository. It is implemented using the generated code skeleton from the API model which is enriched with calls to CDO libraries.

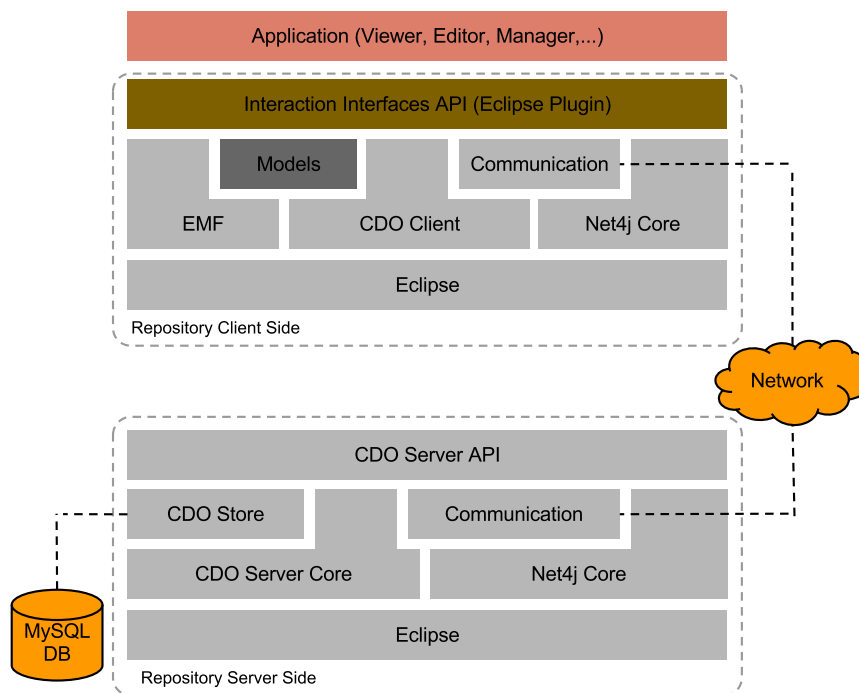


Figure 3.4: Gaya CDO based architecture

## Server

The server part is responsible for managing and storing the data, and provides a set of functionalities to interact with the repository content. As shown in Figure 3.2 (red components), thanks to UML component diagram the server part is composed of the following:

- GayaServer: provides the implementation of the common API,
- GayaMARS: provides the storage mechanisms

The server part of the repository is provided as an Eclipse plugin that will handle the launch of a CDO server defined by a configuration file. This configuration file indicates that a CDO server will be active on a given port and it will make available a CDO repository identified by its name. In addition, the configuration file is used to select which type of database will be used for the proper functioning of the CDO model repository.

## Clients

The client part is responsible for populating the repository and for accessing its content. For this, we identify a set of CDO-based clients as depicted in Figure 3.2. These clients

(turquoise components) provide APIs to applications in order to create the modeling artifacts and in order to use them. For instance, *Gaya4Pattern* and *Gaya4SystemDeveloper* provide a set of APIs for the *Arabion* pattern editor and for the *AccessTool*, respectively.

### 3.2 Repository Interfaces

The repository API is implemented as a CDO client and provided as an Eclipse plugin. The interface methods are defined as a set of operations with input and output parameters. The principles methods with the datatype of the input and output parameters defined in the operation's parameter list are shown in Figure 3.5. The implementation is based on the automatic code generation from the APIs model defined above. The generated Java code defines the different interfaces and functions provided by the repository APIs. The skeleton of the APIs implementations are then completed manually based on CDO technology.

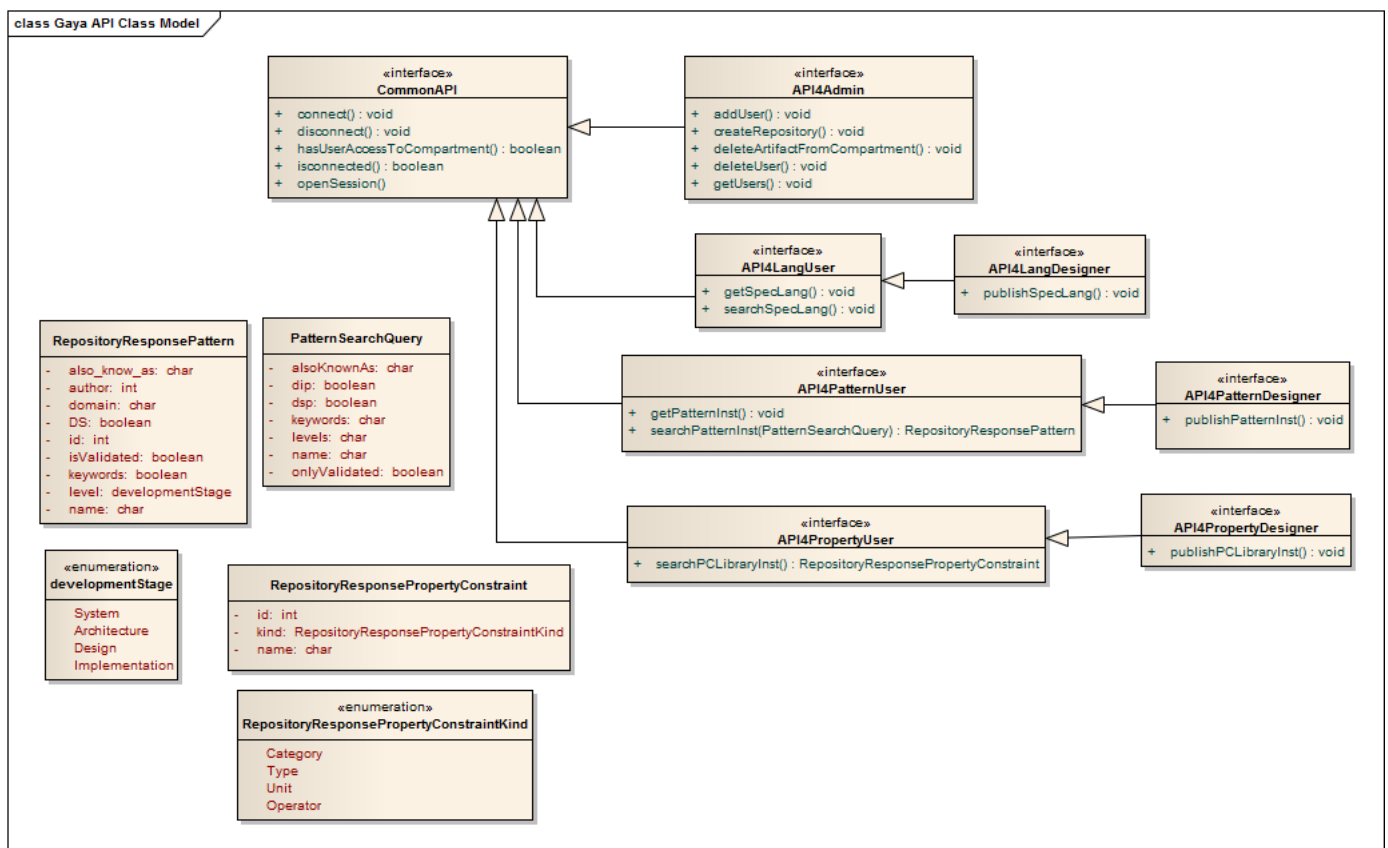


Figure 3.5: The Repository Interfaces and Classes

In the following we review briefly the main APIs and their functions. We provide a Javadoc describing the APIs under <http://www.semcomdt.org/semco/javadoc/>. For example, Figure 3.2.5 visualizes the API for a pattern management.



All Classes

Packages

fr.irit.semco.rc3p.gaya.emare.api4a  
fr.irit.semco.rc3p.gaya.emare.api4admin  
fr.irit.semco.rc3p.gaya.emare.api4n  
fr.irit.semco.rc3p.gaya.emare.api4t  
fr.irit.semco.rc3p.gaya.emare.commonapi

fr.irit.semco.rc3p.gaya.emare.api4a

Classes

API4Pattern  
ArabionPackageProvider  
Gaya4ArabionAPIActivator  
PatternSearchQuery  
RepositoryResponsePattern

Overview Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

fr.irit.semco.rc3p.gaya.emare.api4a

### Class API4Pattern

#### Method Summary

Methods

Modifier and Type	Method and Description
void	<code>getBinaryFile(java.lang.String id, java.lang.String location)</code> <b>Deprecated.</b> <i>Not for public use.</i>
void	<code>getPatternInst(java.lang.String uid, java.lang.String fileName)</code> Gets the Pattern with ID uid from the repository and stores as a file with filename
void	<code>publishPatternInst(java.lang.String patternFileName, boolean force)</code> Publishes a Pattern to the right Compartment.
<code>java.util.ArrayList&lt;RepositoryResponsePattern&gt;</code>	<code>searchPatternInst(PatternSearchQuery query)</code> Searches for Patterns in the repository following the query given as parameter and returns a List of <code>RepositoryResponsePattern</code> which give an overview of the patterns which fit the query.

Methods inherited from class `fr.irit.semco.rc3p.gaya.emare.commonapi.CommonAPI`

`connect`, `disconnect`, `hasUserAccessToCompartment`, `isConnected`, `openSession`

Figure 3.6: API for Pattern

### 3.2.1 Common API

The common part of the SEMCO API regroups all the common actions used by all the APIs like:

- Connecting to the Repository (`connect()`),
- Closing a session (`disconnect()`),
- Checking whether a connection is established (`isConnected()`),
- Checking access to a compartment for a specific User (`hasUserAccessToCompartment()`), ...

### 3.2.2 API for Administration

For management purposes of the repository (see Chapter 4), we provide an API called *API4Admin*, as a specialization of the common API, providing a set of functions, which are:

- User management (*addUser()*, *deletUser()*, *getUsers()*),
- Repository management (*createRepository()*) and
- Compartment management (*addUser()*, *addUser()*, ...)

The *API4Admin* API is used by the repository management tool presented in Chapter 4.

### 3.2.3 API for Languages

For the management of the specification languages, we provide an API called *API4Lang*, which specializes the common one, providing the following methods:

- Publishing (*publishSpecLang()*),
- Searching (*searchSpecLang()*),
- Instantiation (*getSpecLang()*).

### 3.2.4 API for Property

For the interaction with the repository when targeting property libraries, we provide an API called *API4Property*, which specializes the common one. The interface methods are:

- Publishing (*publishPCLibraryInst()*),
- Searching (*searchPCLibraryInst()*),
- Instantiation (*getLibraryInst()*).

The results of the search request is a list of property libraries (*RepositoryResponsePropertyConstraint*) fulfilling the search query. The list may be splitted on several libraries (*RepositoryResponsePropertyConstraintKind*). Instantiating a property library for the editor will then yield consistency checking for automatic validation of model dependencies. For example, category library can be instantiated, where a missing of a resource (unit or type library) will yield an error message. We manage these errors using the Java Exception mechanisms. We define an exception named *RepositoryDependencyException* which is thrown when problems occur with missing local dependencies.

The *API4Property* API is used by the Deposit and Retrieval tools presented in Section 5.1.2 and in Section 5.3.1, respectively.

### 3.2.5 API for Pattern

For the management of patterns, we provide an API called *API4Pattern* specializing the common one (see Figure 3.2.5) and offering a set of functions, which are:

- Publishing (*publishPatternInst()*),
- Searching (*searchPatternInst()*),
- Instantiation (*getPatternInst()*)...

The results of the search request (*PatternSearchQuery*) is a list of patterns (*RepositoryResponsePattern*) fitting the search query. The structure of the search query and the response are shown in Figure 3.5.

Instantiating a pattern for the editor will then yield consistency checking for automatic validation of model dependencies. For example, a pattern cannot be instantiated, when there is a resource (category library) missing. It will yield an error message, as visualized in Figure 5.19. We manage these errors using the Java Exception mechanism. We define an exception named *RepositoryDependencyException* which is thrown when problems occur with missing local dependencies with the required property libraries.

The *API4Pattern* API is used by the Deposit and Retrieval tools presented in Section 5.2.5 and in Section 5.3.2, respectively.

## 3.3 Gaya Product

The Gaya tool is provided as an Eclipse Plugin, based on the Eclipse Modeling Framework Technologies (EMFT). We provide an installation based on the Eclipse standards of the *p2-repository (update-site)*.

### 3.3.1 Installation

The current version is installable via the installation routines of the Eclipse Platform and our update-site<sup>3</sup>. The process of installation is the as follows:

1. Install *Acceleo* via the Eclipse Marketplace
  - Help > Eclipse Marketplace
  - Search for *Acceleo*
  - Install
  - Restart
2. Add the update-site to the *Available Software Sites*

<sup>3</sup><http://www.semcomdt.org/semco/tools/updates/1.2>

- Window > Preferences
- Install/Update > Available Software Sites
- Add...
- Add *SEMCO* as name and add the URL

### 3. Install Gaya

- Help > Install New Software
- Work With: Select *SEMCO*
- Check Gaya
- Next
- Next and accept the License
- Finish
- Restart

#### 3.3.2 Gaya License

The Gaya Tool, like the whole Tool Suite SEMCO is licensed under the EPL, as it is the most common license for products built on the Eclipse Platform or using its technologies. It is a license which, on one hand, is designed to be business-friendly and, on the other hand, to be compliant with OSI's (Open Source Initiative<sup>4</sup>) and FSF's (Free Software Foundation<sup>5</sup>) understanding of *Free Software*.

---

<sup>4</sup><http://opensource.org/>

<sup>5</sup><http://www.fsf.org/>

## 4 Repository Management

Repository management is implemented via the *GayaAdmin* tool. *GayaAdmin* offers repository management with facilities such as user management, compartment management, property management and pattern management. We offer these facilities through a set of dialogues triggered in the *GayaAdmin* tool. The main dialogue is shown in Figure 4.1.

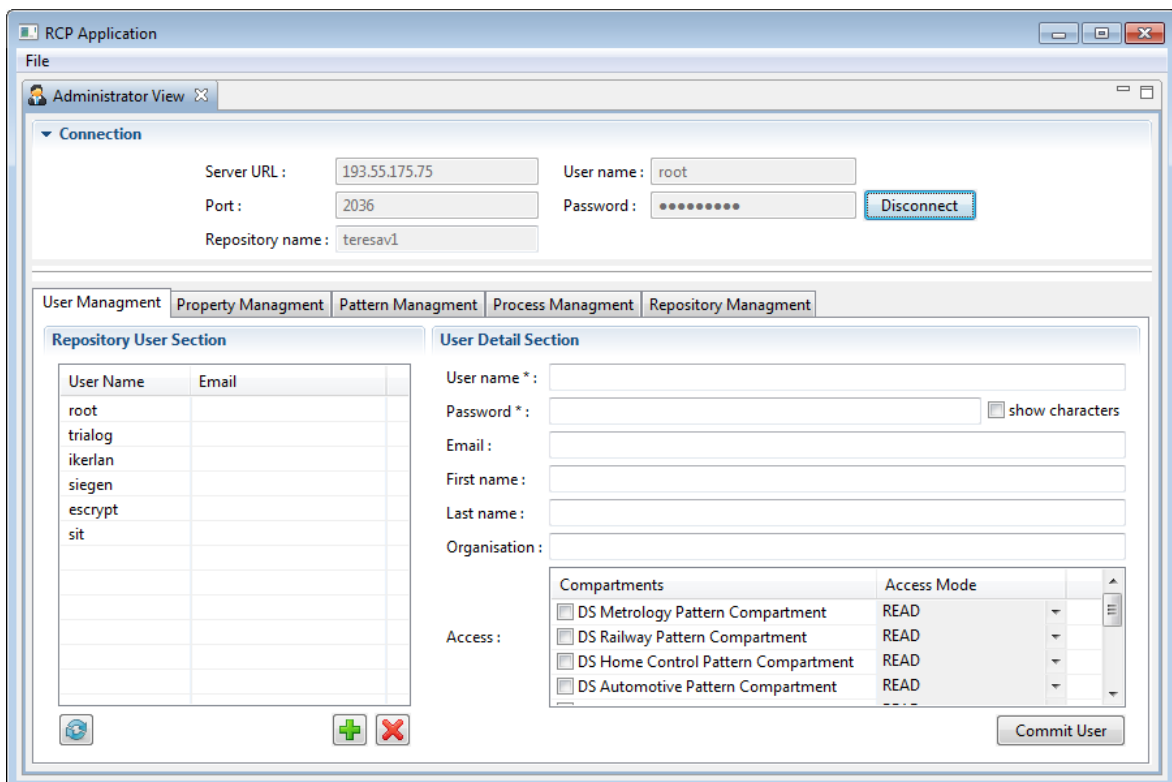


Figure 4.1: The Admin UI of the Repository

## 4.1 User Management

The user management supports a set of features such as user lookup, add, removal, sorting and categorization as shown in in the left part of Figure 4.2. The right part shows the necessary information for adding a new user. We need to provide the necessary details, which are the username, password, affiliation, email and organization to create user instances. Then, we specify the user access mode (RW) per compartment.

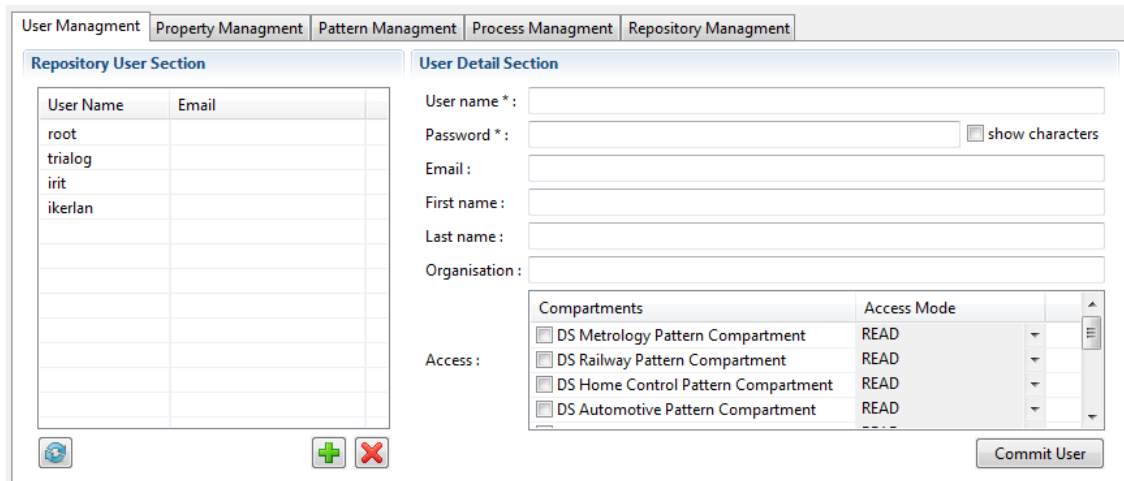


Figure 4.2: User Management Part

## 4.2 Compartment Management

The compartment management offers repository management with facilities such as compartment add, removal and sorting. We offer these facilities through a dialogue shown in Figure 4.3. There is a list of compartment on the left and a compartment identification information view on the right.

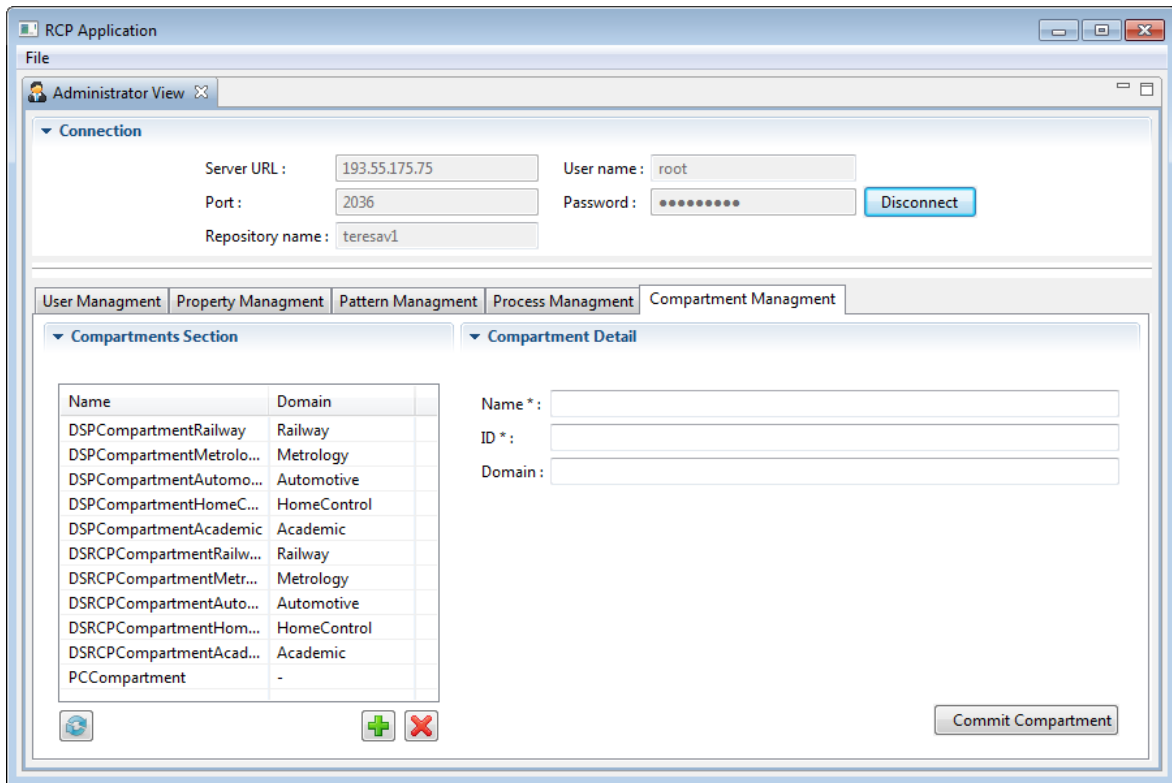


Figure 4.3: Compartment Management

## 4.3 Authentication

The user authentication dialogue is visualized in Figure 4.4. The authentication allows the user to access to the repository resources regarding its credentials. The user has to provide its name and password, the repository name and the repository location.

- In the host field type the address of the host (URI). In our case `http://www.semcomdt.org` which is hosted by the University of Toulouse 2.
- In the Repository name field, type the name of the repository on the host ( for TERESA `teresav1`).
- In the User field, type the username under which you want to connect to the repository.
- In the Password field, type the password for the above username.

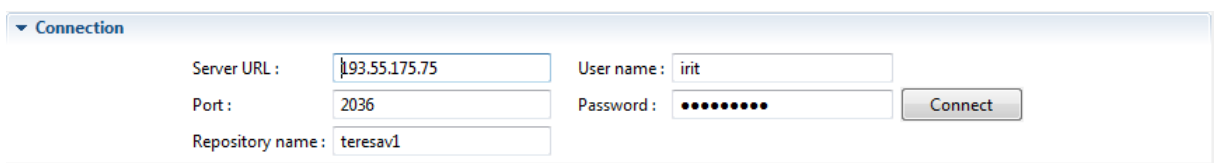


Figure 4.4: Repository Authentication

As the toolsuite is provided as Eclipse plugins, we provide also an authentication facility as an Eclipse preferences set as shown in Figure 4.5.

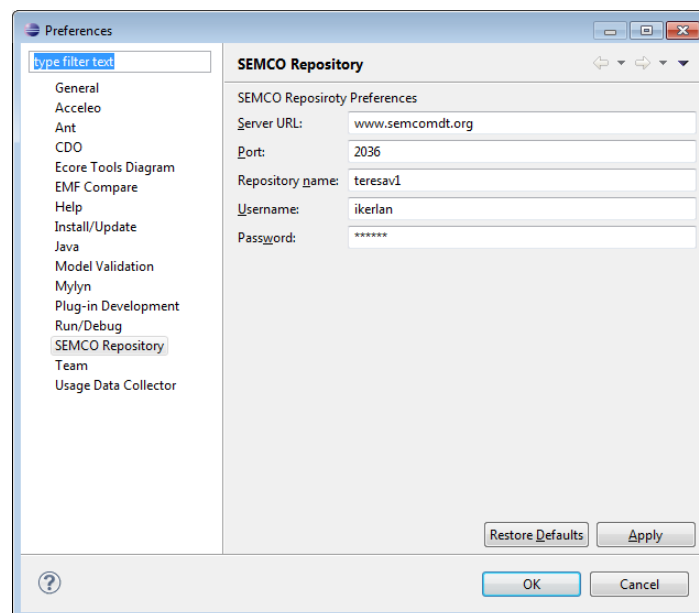


Figure 4.5: Repository Authentication Under Eclipse



## 4.4 Property Management

The property management supports a set of features such as property library lookup, removal, sorting, exporting and categorization as shown in Figure 4.6.

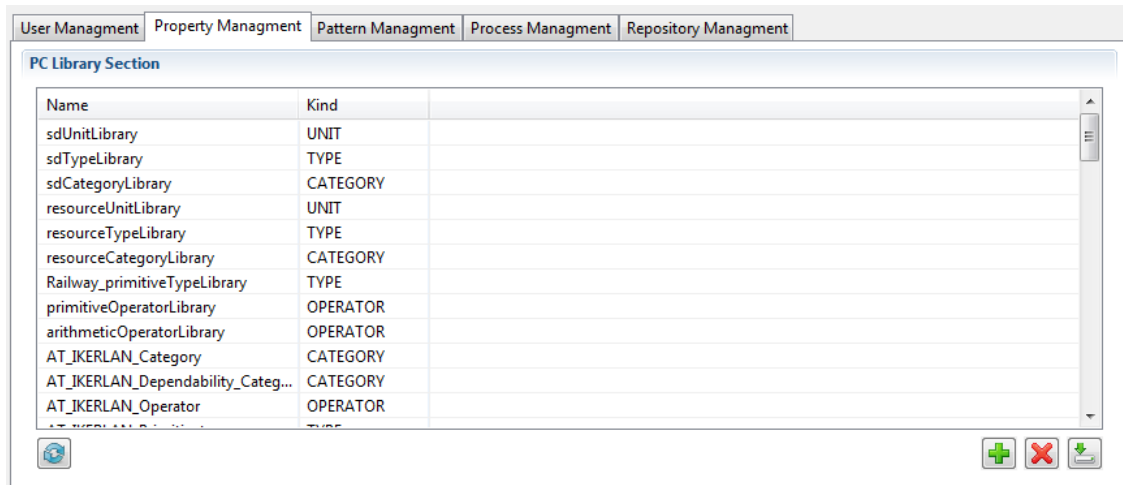


Figure 4.6: Property management part

## 4.5 Pattern Management

The pattern management supports a set of features such as pattern lookup, removal, sorting, exporting and categorization as shown in Figure 4.7.

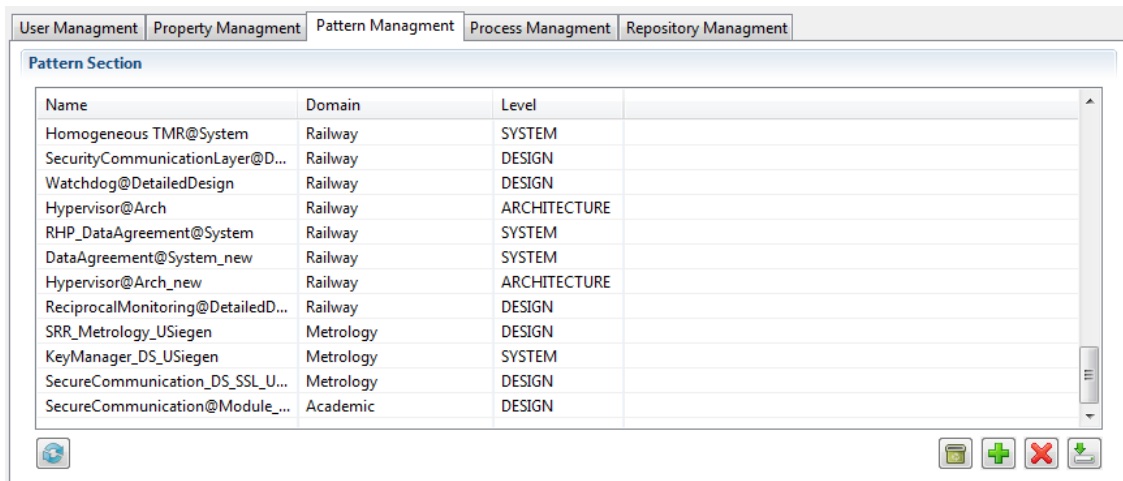


Figure 4.7: Pattern management part

## 5 Repository Populating

Here we present our tools for populating the repository. As we shall see, the following sections depict the new environment for designing resource and S&D properties libraries and S&D patterns.

### 5.1 Property Designer

To create a property model, *Tiqueo* implements several facilities conforming to the PCM metamodel to manage property libraries including units, types, categories and the different expressions of constraints on these properties. *Tiqueo* supports a number of features such as design of a property library, validation, deposit and retrieval into and from the repository.

The main process model for building property library is visualized in Figure 5.1, pointing three principal activities:

- Create a Unit Library,
- Create a Type Library,
- Create a Category Library.

Once the appropriate activity is selected, the left and right part of Figure 5.2 show the process for building a Type Library and a Category Library, respectively.

For a category library, the design environment is presented in Figure 5.3. There is a design palette on the right<sup>1</sup>, a tree view of the project on the left and the main design view in the middle. Category library is built using types libraries instances. In our example, an instance of the *sdTypeLibrary* called *sdTypeLibrary.tm* is imported from the repository to the local project workspace using the *Retrieve* tool (see Section 5.3.1). Then, the user has to create a reference to this library as a resource, such as illustrated in Figure 5.4.

These libraries are used as external model libraries to type the properties of the patterns. Especially during the editing of the pattern (see next section) we define the properties and the constraints using these libraries.

---

<sup>1</sup>(1) for unit, (2) for type and (3) for category.

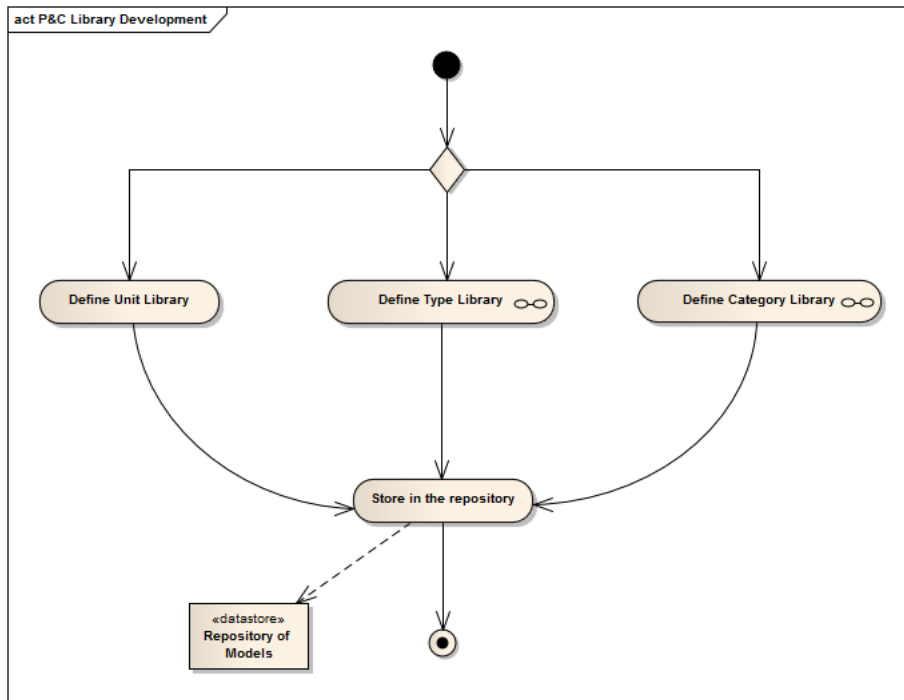


Figure 5.1: P&C Library Development

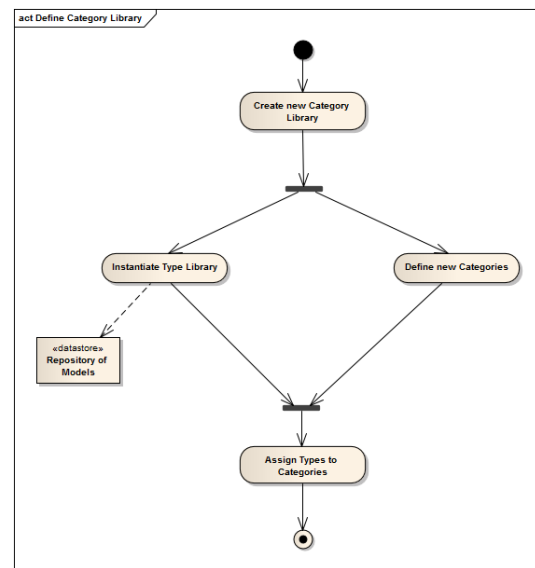
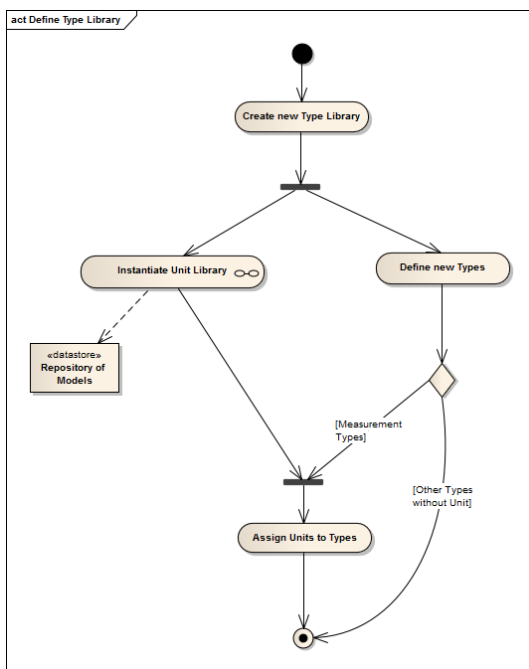


Figure 5.2: Type and Category Libraries Definition Processes

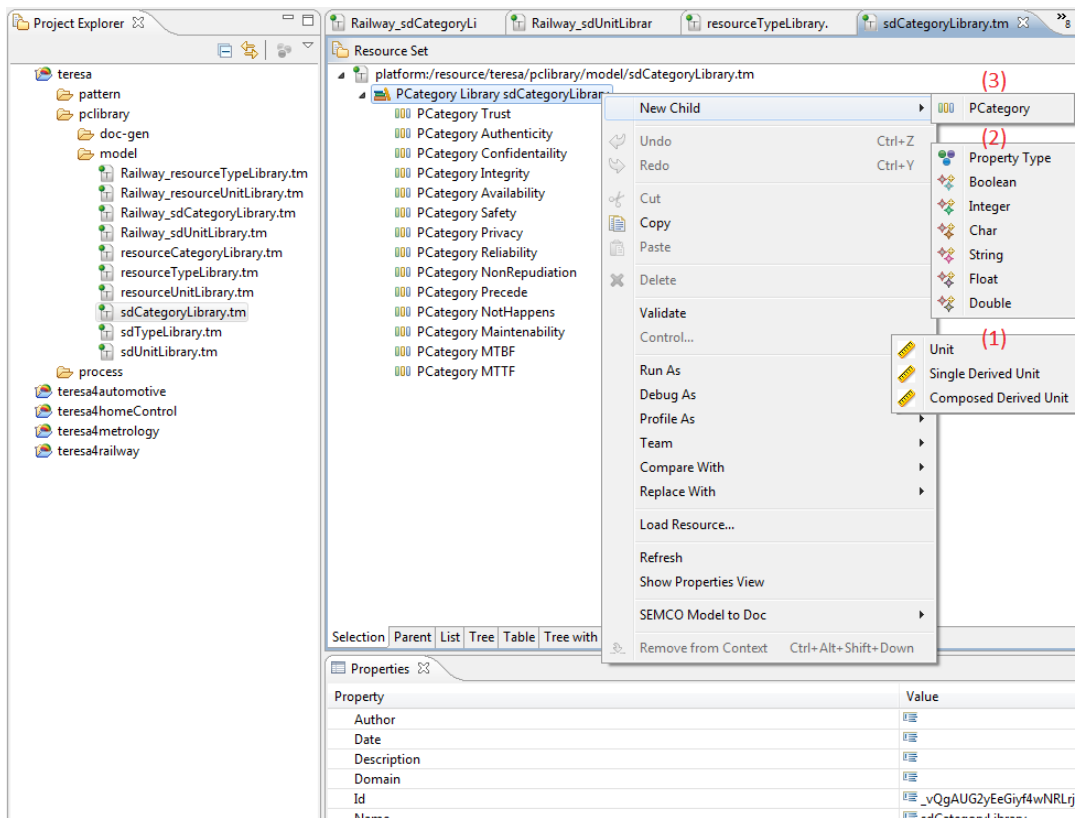


Figure 5.3: Designing a Category Library

### 5.1.1 Property Validation

The property validation tool is used to guarantee design validity conforming to the property metamodel as visualized in Figure 5.5. Property model validation starts by right clicking on Property Core and pressing the *Validation* tool.

### 5.1.2 Property Deposit

Property library publication is triggered by running the *Publication* tool by right clicking on Property Core and pressing the *Publication* tool. The deposit tool requires the execution of the validation tool to guarantee design validity. When executed, as shown in Figure 5.6, the library will be stored in the repository. The tool uses the *Gaya4Property* API (see Section 3.2.4) for the deposit into the repository.

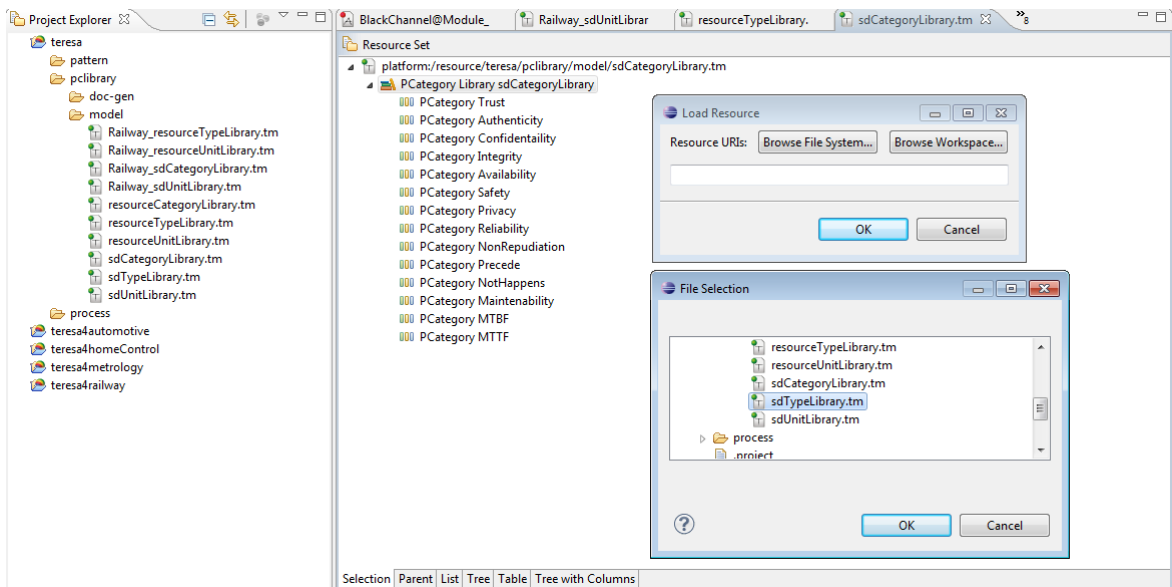


Figure 5.4: Eclipse Load Resource Tool

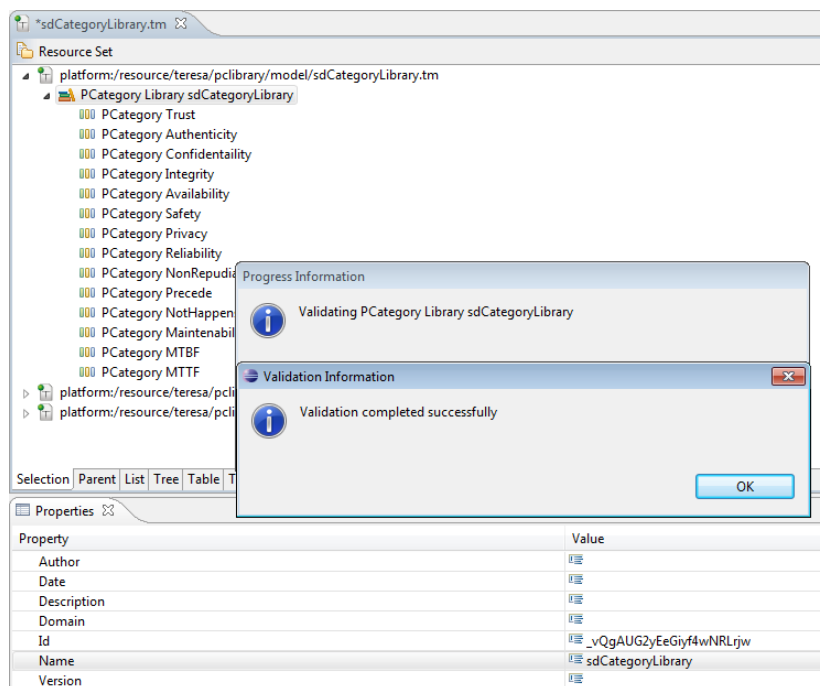


Figure 5.5: Property Validation

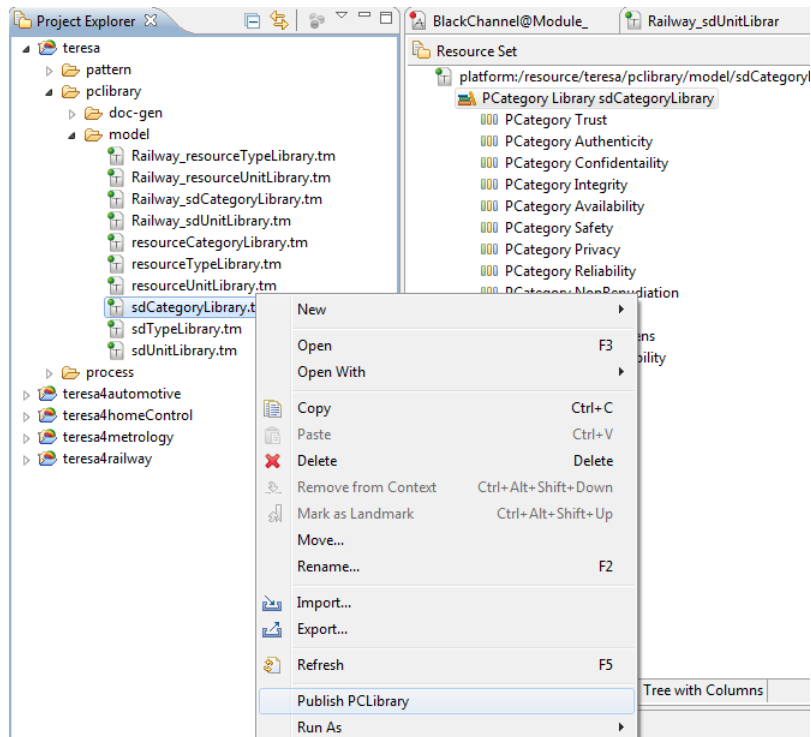


Figure 5.6: Property Deposit

### 5.1.3 Tiqueo Product

The Tiqueo tool is provided as an Eclipse Plugin, based on the Eclipse Modeling Framework Technologies (EMFT). We provide an installation based on the Eclipse standards of the *p2-repository (update-site)*. The current version is installable via the installation routines of the Eclipse Platform and our update-site<sup>2</sup>. The process of installation is the as follows:

1. Install *Acceleo* via the Eclipse Marketplace
  - Help > Eclipse Marketplace
  - Search for *Acceleo*
  - Install
  - Restart
2. Add the update-site to the *Available Software Sites*
  - Window > Preferences
  - Install/Update > Available Software Sites

<sup>2</sup><http://www.semcomdt.org/semco/tools/updates/1.2>

- Add...
- Add *SEMCO* as name and add the URL

### 3. Install Tiqueo

- Help > Install New Software
- Work With: Select *SEMCO*
- Check Tiqueo
- Next
- Next and accept the License
- Finish
- Restart

#### 5.1.4 Tiqueo License

The Tiqueo Tool, like the whole Tool Suite SEMCO is licensed under the EPL, as it is the most common license for products built on the Eclipse Platform or using its technologies. It is a license which, on one hand, is designed to be business-friendly and, on the other hand, to be compliant with OSI's (Open Source Initiative<sup>3</sup>) and FSF's (Free Software Foundation<sup>4</sup>) understanding of *Free Software*.

---

<sup>3</sup><http://opensource.org/>

<sup>4</sup><http://www.fsf.org/>

## 5.2 Pattern Designer

The pattern designer called *Arabion* supports a number of features including pattern design at DIPM and DSPM level, validation, interaction with a verification framework (see WP5), deposit and retrieval into and from the repository, respectively.

The process root, as shown in Figure 5.7, indicates the start of the creation of an DIPM pattern interacting with *Pattern Repository*<sup>5</sup>, a data base of informal definitions of patterns. It contains some initialization actions to define the pattern attributes (e.g, name, author, date, . . .). The next activities are the following ones:

- *Keyword*. It defines a set of keywords to ease the search of the pattern.
- *Define internal structure*. It implements the static and the dynamic representation of the solution. The activity is achieved using external tools (e.g, Papyrus UML Modeler tool).
- *Develop external interfaces*. It defines the exposed interfaces as a set of operations.

The next concern of the process is the definition of the pattern properties and constraints. The supporting activities require the interaction with the *Model Based Repository* in order to instantiate the property libraries. The invoked activities are the following ones:

- *Search*.
- *Select*.
- *Import*.

After the instantiation of the appropriate libraries, one resource is created for each library. This resource remains active for the complete duration of the process. The imported model libraries will be used during the definition of the properties to type their category.

The next activity deals with the pattern validation. It supports the formal validation of a pattern using an external process. The result is a set of validation artifacts. At this point, the pattern designer may generate documentation. If the pattern has been correctly defined, i.e conforms the pattern metamodel, the pattern is ready for the publication into the model-based repository.

### 5.2.1 Modeling Pattern at DIPM

For an DIPM pattern, the design environment is presented in Figure 5.8. In this instance a DI pattern called *SecureCommunication@Module* was designed. The main view shows that *SecureCommunication@Module* is a DI pattern built by specifying a set of properties, interfaces and an internal structure. Each property has a category typed with a property library as shown in the properties box. The pattern designer has to provide the necessary information to define a property, mainly the name and the description as textual fields. The

---

<sup>5</sup>Informal description of the pattern, such as the one used in the appendix document of the Deliverable D4.2.



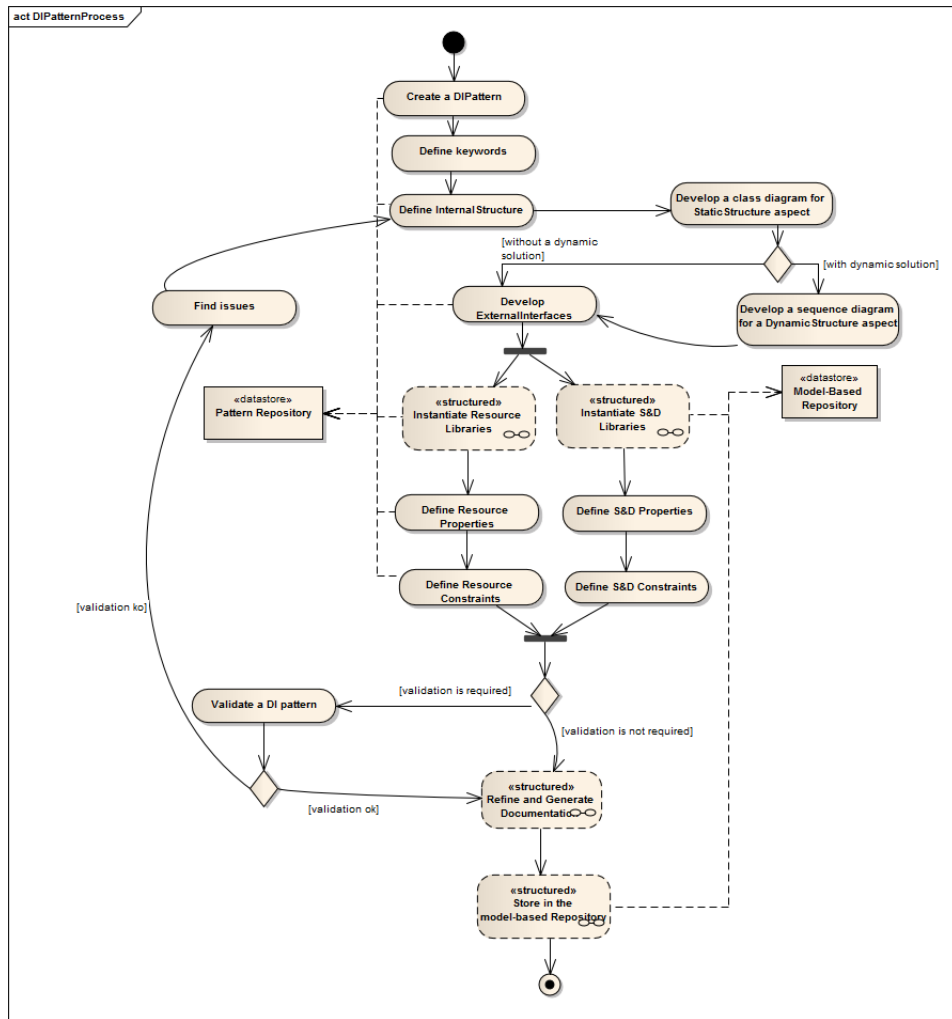


Figure 5.7: Pattern development process at DIPM

internal structure was specified using UML diagrams edited by an external UML editor. Interfaces are defined with respect to the pattern metamodel. The pattern has interfaces containing the provided service. In our case *sender* and *receiver* with a set of operations: *send* and *receive*, which takes a set of inputs and produce a set of outputs. The data sender Channel Authentication has its own authentication Key that identifies itself in the communication. The data receiver knows this key and uses it to authenticate each message coming from the communication layer. When authentication process successfully completed, it will let the message pass over to the receiver application. The key of the sender will be correctly codified in order to avoid an external attacker to know it and impersonate the sender, sending malicious information to the receiver.

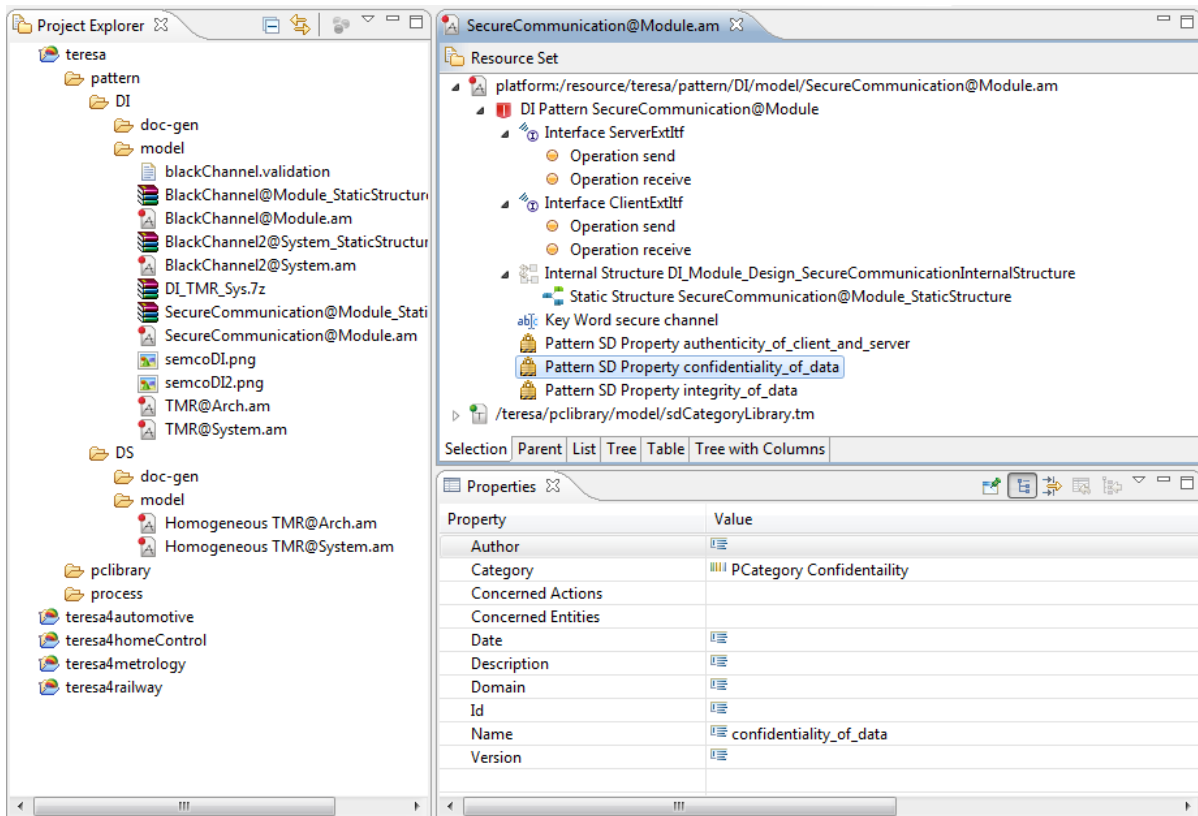


Figure 5.8: Secure Communication DI Pattern at Design level

## 5.2.2 Modeling Pattern at DSPM

For a DSPM pattern, the design environment is presented in Figure 5.9. There is a design palette on the right, a tree view of the project on the left and the main design view in the middle. The design palette is updated regarding the one used for a DIPM pattern to display suitable design entities for building pattern at DSPM. These entities are *internal interfaces* and *resource properties*. There are also other design entities such as *domain* and *refinement*. DS patterns are built by refining DI pattern. In our example, the *SecurityCommunicationLayer@DetailedDesign* pattern refines the *SecureCommunication@Module* pattern for the railway domain using *HMAC* mechanism. Like DI patterns, the DS pattern has external interfaces.

Having defined the internal structure by refining the DI's one using a specific mechanism, we also had to specify the internal interfaces, the S&D and the resource properties. The resource property is specified using the same activities as those concerning the S&D property.

As is described in the Domain Independent manner, the data sender has an identification key, which in this case will be encoded together with the data that will be sent by an HMAC algorithm to elude any malicious intention to extract the sender's key. The call of

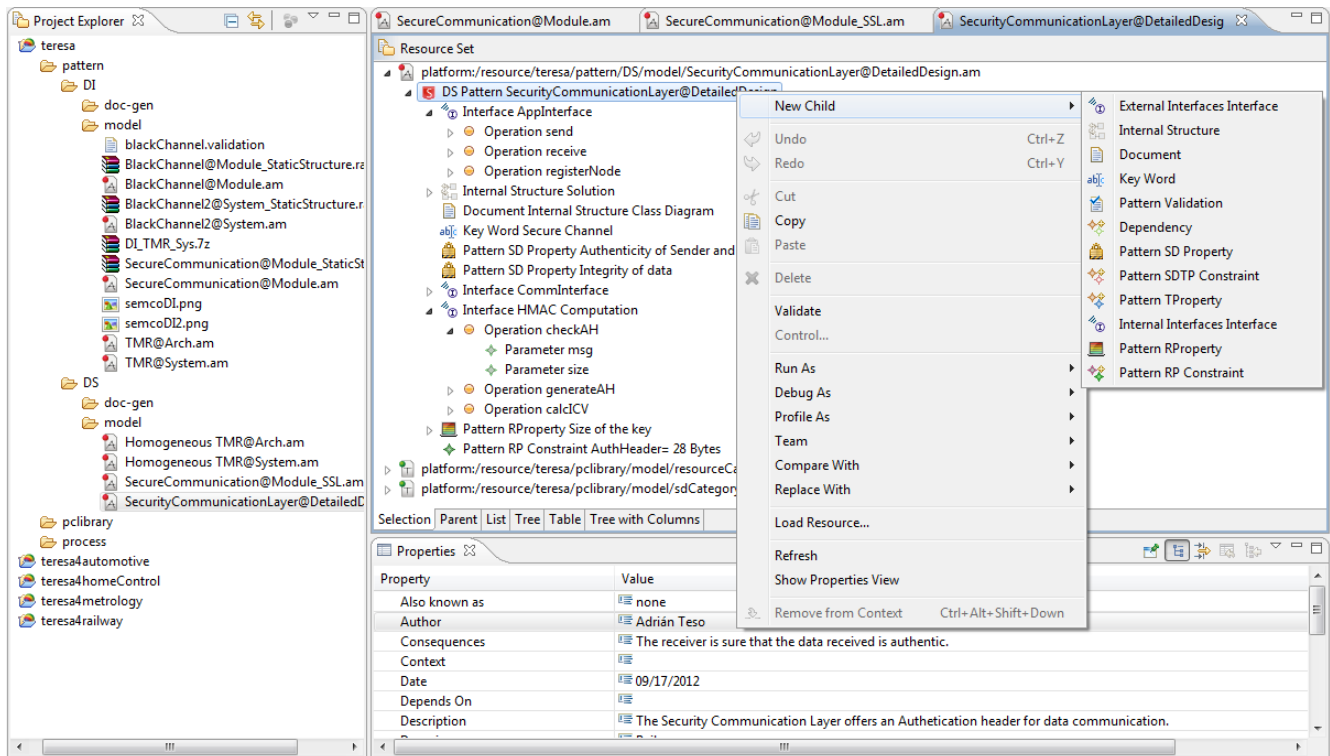


Figure 5.9: Secure Communication DS pattern at Design level

the method *send()* of the Sender calls internally to *generateAH()* to prepare an appropriate authentication header for the data. Once this header is appended to the message it is sent by the communication channel. On the Receiver's side, the call of the method *receive()* returns the last received message from the sender. This message is checked by the method *checkAH()*. If the message is correct it is passed to the application, in any other case is discarded. The operations *generateAH()* and *checkAH()* are provided through an internal interface called *HMAC Computation*.

To type the category of an S&D property, the user has to create a reference to the library as a resource, as illustrated in Figure 5.10. As a prerequisite, the designer uses the *Retrieval* tool (see Section 5.3.1) to search and then to upload the appropriate library in its environment.

In our example, an instance of the *sdLibrary* called *sdCategoryLibrary.tm* is imported from the repository to the local project workspace. We specified an S&D property called *Authenticity of Sender and Receiver*. To type the category of this property, we use the one defined in the library: *Authenticity*.

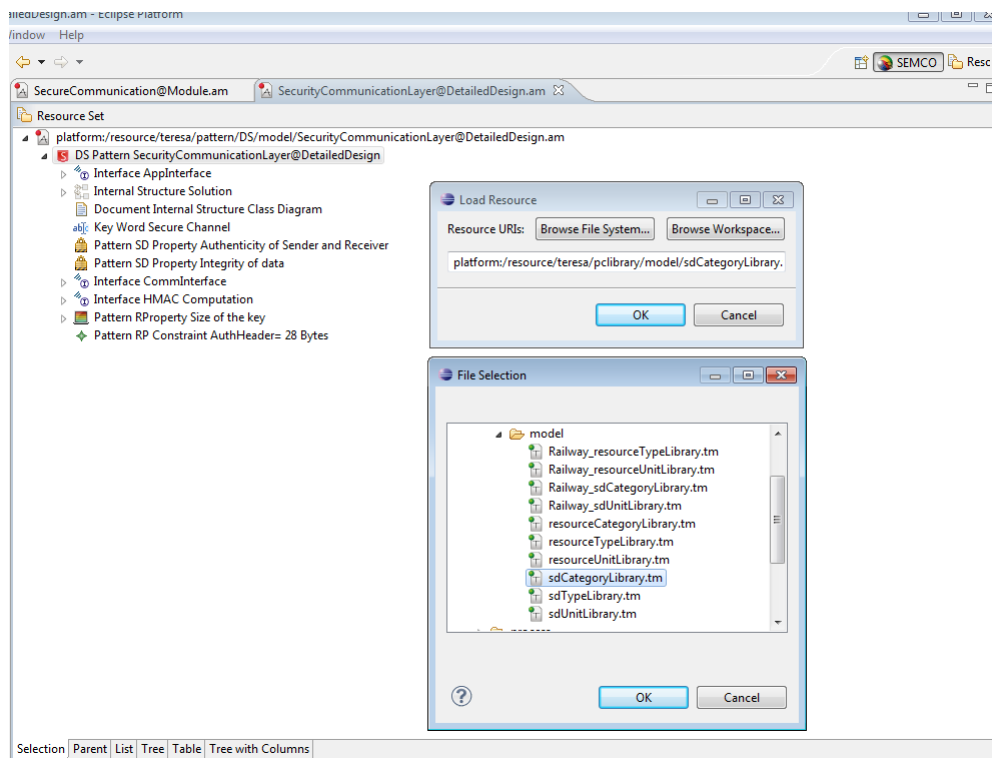


Figure 5.10: Eclipse Load Resource Tool

### 5.2.3 Pattern Conformance Validation

The pattern validation tool is used to guarantee design validity conforming to the pattern metamodel. Pattern model validation starts by right clicking on Pattern Core and pressing the *Validation* tool. In our example, Secure communication pattern model can be validated, where a violation of a metamodel construct will yield an error message (see Figure 5.11).

### 5.2.4 Generation of Documentation

Documentation generation of a pattern model is triggered by running the *SEMCO model to Doc* tool, as visualized in Figure 5.12. The tool starts by right clicking on Pattern Core and pressing this tool. Our implementation so far allows to generate HTML document using Model to Text model transformation through Acceleo<sup>6</sup>. For example, Figure 5.13 visualizes the results of the transformation.

<sup>6</sup>[www.acceleo.org/](http://www.acceleo.org/)

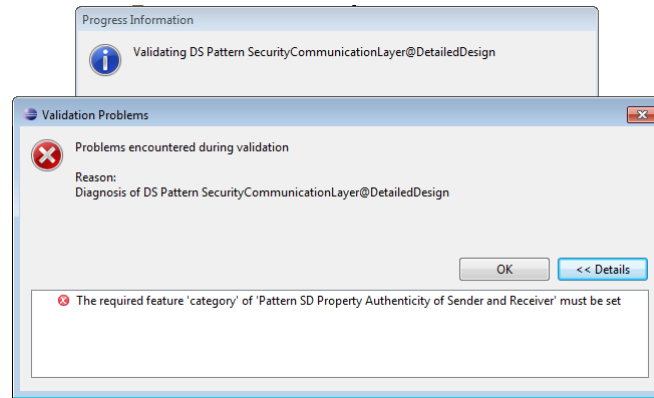


Figure 5.11: Pattern Validation

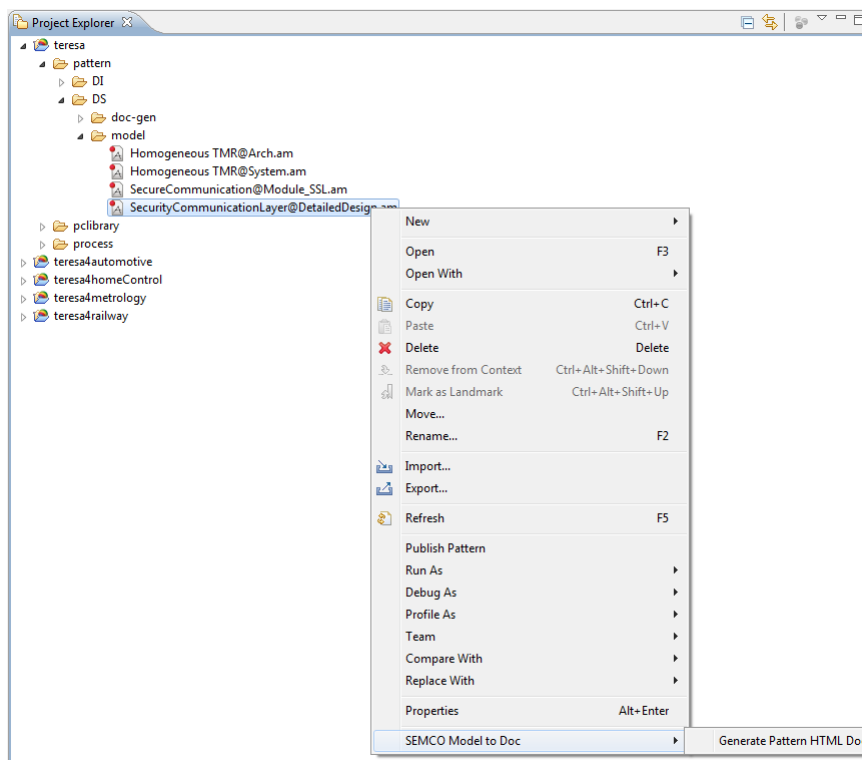


Figure 5.12: Generation of Documentation

### 5.2.5 Pattern Deposit

Pattern publication is triggered by running the *Publication* tool, as visualized in Figure 5.14. The tool starts by right clicking on Pattern Core and pressing the *Publication* tool. When executed, the pattern will be stored in the repository following the pattern designer's profile (compartment). The tool uses the *Gaya4Pattern* (see Section 3.2.5) for publishing to the

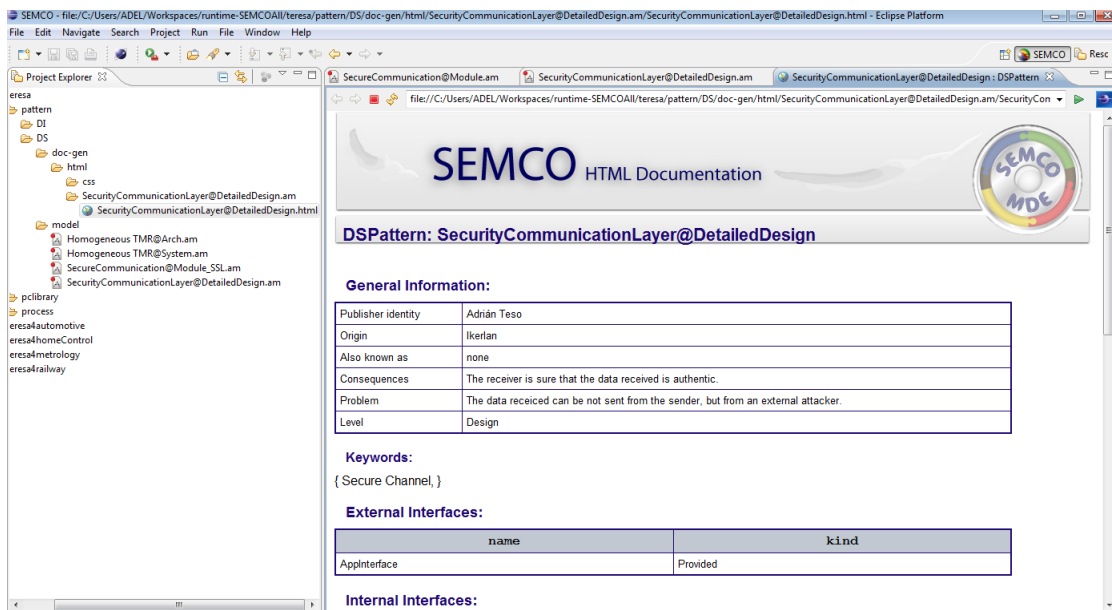


Figure 5.13: HTML Documentation

repository. Note, however that the deposit tool requires the execution of the validation tool to guarantee design validity.

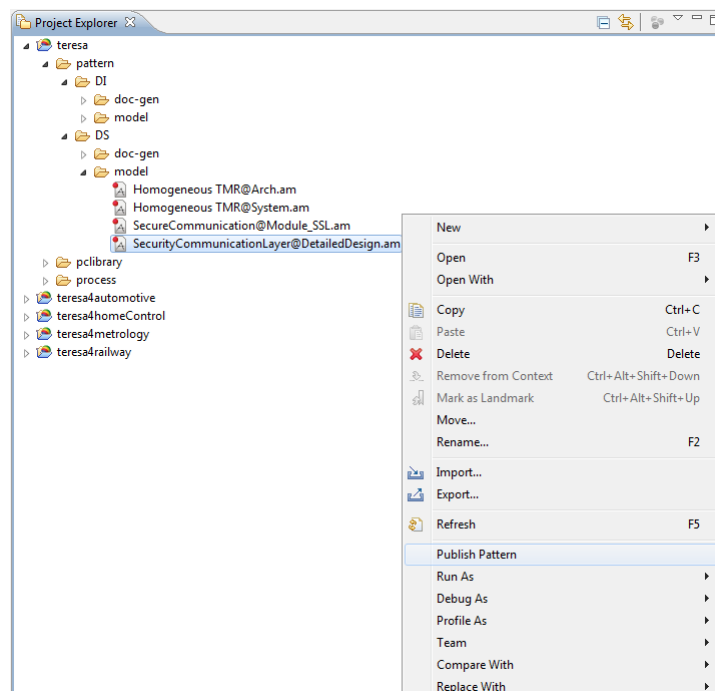


Figure 5.14: Pattern Publication

## 5.2.6 Validation Artifacts

In the current version, we propose to use the SIT plugin (see WP5) to edit a property (or a set of properties) and then to store the result as a *xSeMF* model. This model is then attached to the pattern using the existing design entity called *Pattern Validation*, as visualized in Figure 5.15. The references among the pattern properties and the *xSeMF* model will be done by conventions as shown in Figure 5.16. In the example, we use the SIT tool to specify the authenticity property of the secure communication pattern. The result is a pattern validation artifact called *xsemf Authenticity*, stored in the file named *prop\_auth.xsemf* (Figure 5.15). Then, we refer to this validation artifact during the specification of the S&D property *authenticity of Sender and Receiver* such as shown in Figure 5.16.

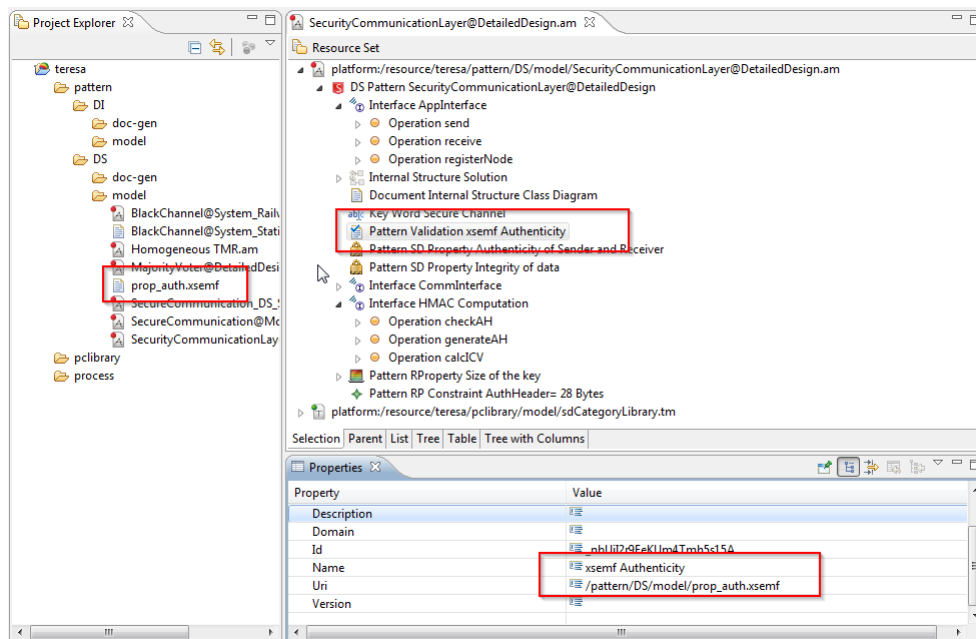


Figure 5.15: Pattern Validation Artifact -a

## 5.2.7 Arabion Product

The Arabion tool is provided as an Eclipse Plugin, based on the Eclipse Modeling Framework Technologies (EMFT). We provide an installation based on the Eclipse standards of the *p2-repository (update-site)*. The current version is installable via the installation routines of the Eclipse Platform and our update-site<sup>7</sup>. The process of installation is the as follows:

1. Install *Acceleo* via the Eclipse Marketplace
  - Help > Eclipse Marketplace

<sup>7</sup><http://www.semcomdt.org/semco/tools/updates/1.2>

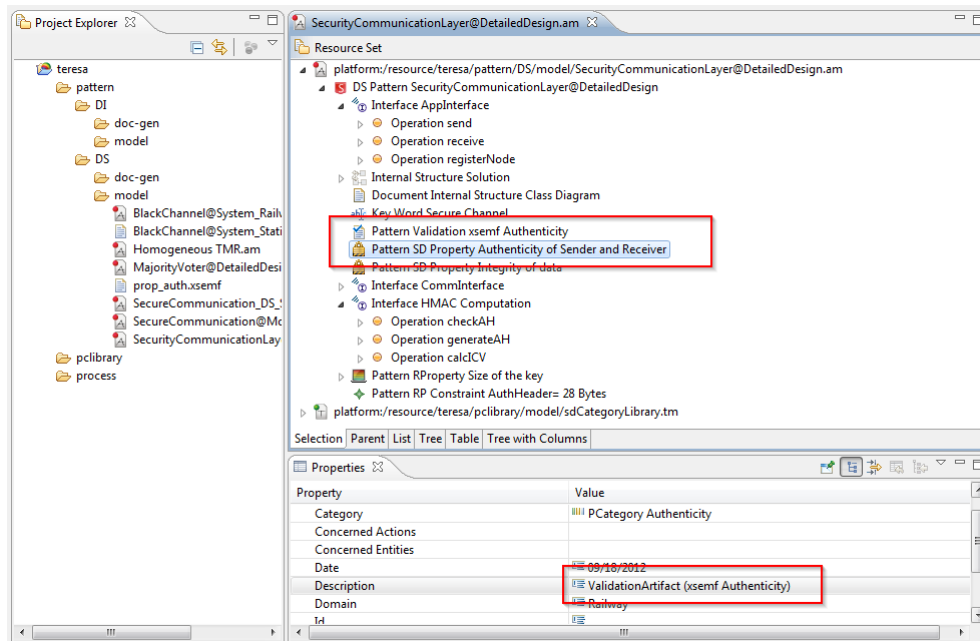


Figure 5.16: Pattern Validation Artifact -b

- Search for *Acceleo*
  - Install
  - Restart
2. Add the update-site to the *Available Software Sites*
    - Window > Preferences
    - Install/Update > Available Software Sites
    - Add...
    - Add *SEMCO* as name and add the URL
  3. Install Arabion
    - Help > Install New Software
    - Work With: Select *SEMCO*
    - Check Arabion
    - Next
    - Next and accept the License
    - Finish
    - Restart



### 5.2.8 Arabion License

The Arabion Tool, like the whole Tool Suite SEMCO is licensed under the EPL, as it is the most common license for products built on the Eclipse Platform or using its technologies. It is a license which, on one hand, is designed to be business-friendly and, on the other hand, to be compliant with OSI's (Open Source Initiative<sup>8</sup>) and FSF's (Free Software Foundation<sup>9</sup>) understanding of *Free Software*.

---

<sup>8</sup><http://opensource.org/>

<sup>9</sup><http://www.fsf.org/>

## 5.3 Repository Retrieval

In this part, we present the TERESA repository tools for the assistance of modeling artifacts development process.

### 5.3.1 Property Retrieval

As mentioned before, when building a pattern we use properties libraries to type its properties. The tool provides a chain of library selection and instantiation dialogues. The library search/selection dialogue is shown in the right part of Figure 5.17. The tool uses the *Gaya4Property* API for the search/selection of the property library which is used during a pattern and a property library development processes.

The results are displayed in search results tree as Unit Library, Type Library, Category Library and Operator Library. For example, the right part of Figures 5.17 shows that there is one category library called *sdCategoryLibrary* published in the repository with keyword *confidentiality*. In addition, the Tool includes features for exportation and instantiation as dialogues. So once selected, we need to provide the necessary details, which are project path and instance name, as visualized in the left part of Figures 5.17.

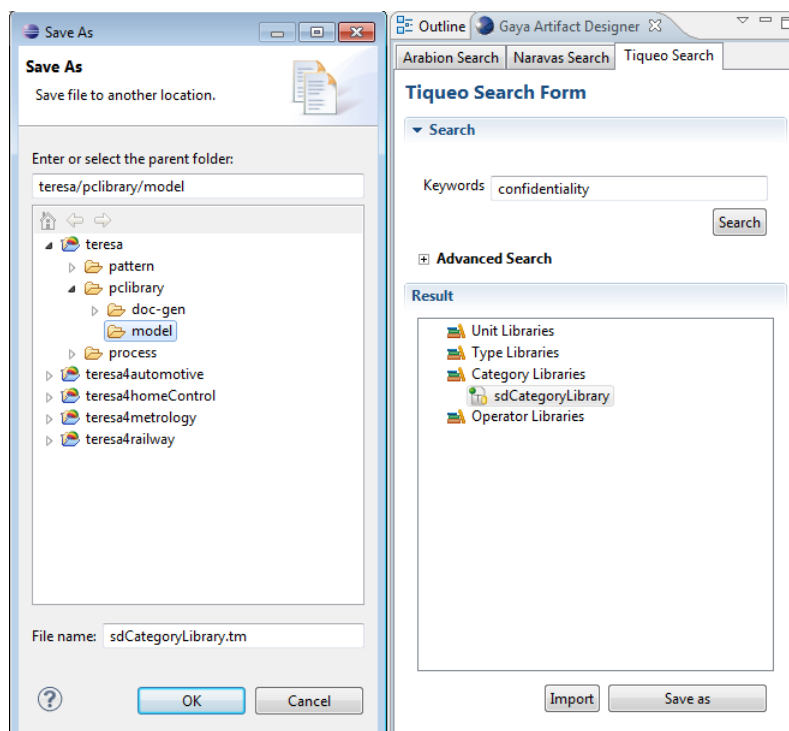


Figure 5.17: Property Library Instantiation

### 5.3.2 Pattern Retrieval

As mentioned before, DSPM patterns are built from DIPM patterns. The tool uses the *Gaya4Pattern* API for the search/selection of the patterns which is used during a pattern and a system development process. For instance, as shown in the right part of Figure 5.18, the tool provides the following facilities to help the selection of appropriate patterns regarding:

- key words,
- domain independent vs. domain specific,
- lifecycle stage,
- S&D categories,
- resources categories.

The results are displayed in search results tree as System, Architecture, Design and Implementation patterns. For example, the right part of Figures 5.18 shows that there is a DI pattern at design level targeting the *Confidentiality* S&D property<sup>10</sup>, named communication and has a keyword *secure*. The Tool includes also features for exportation and instantiation as dialogues. In our case, we select the *Secure Communication* pattern for instantiation providing the necessary information, including the project path and instance name (see the left part of Figures 5.18). The result can be used to design a DSPM pattern, as presented above. In addition, the tool includes a dependency checking mechanism. For example, a pattern can be instantiated, where a missing of a resource (property library) will yield an error message (see Figure 5.19).

---

<sup>10</sup>In our modeling, this means that the pattern has a property with a confidentiality category type.

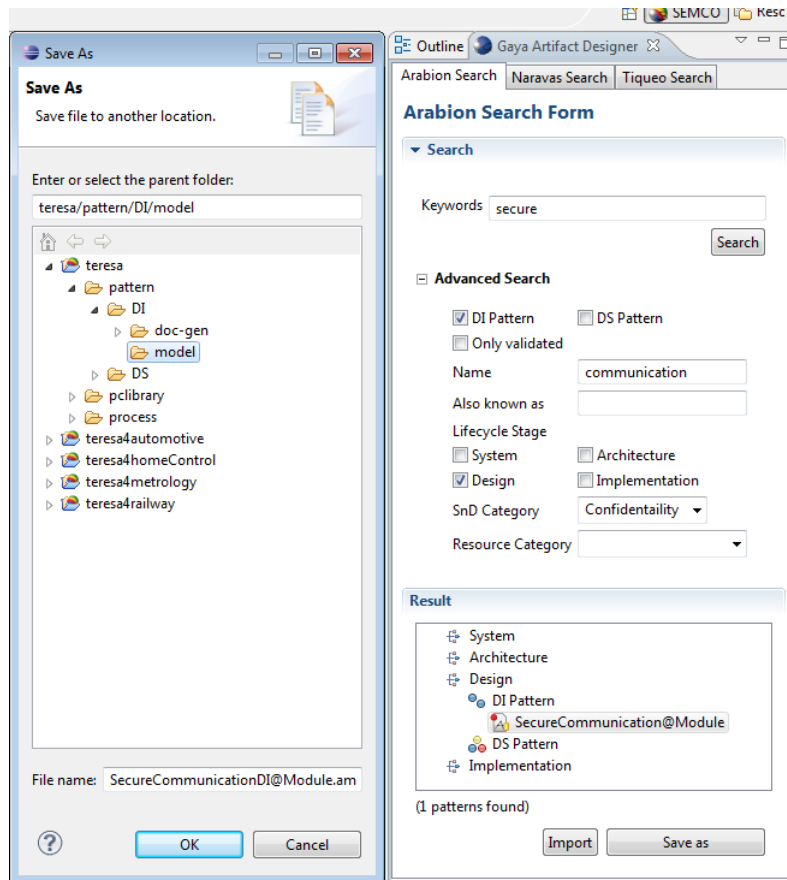


Figure 5.18: Pattern Instantiation

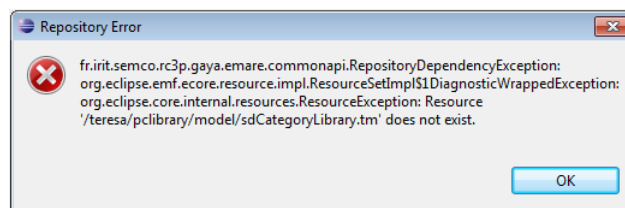


Figure 5.19: Pattern Instantiation - Consistency

## 5.4 An Overview of the TERESA Repository Content

The TERESA repository contains so far (on January 2013):

- *Property Libraries*. 69 property model libraries (see the left part of Figure 5.20):
  - 12 Unit Libraries
  - 23 Type Libraries
  - 20 Category Libraries
  - 14 Operator Libraries
- *Pattern Libraries*. 59 patterns (see the right part of Figure 5.20):
  - 20 System Level patterns (12 DI, 8 DS)
  - 25 Architecture Level patterns (9 DI, 16 DS)
  - 14 Design Level patterns (3 DI, 11 DS)
  - 0 Implementation Level patterns (0 DI, 0 DS)

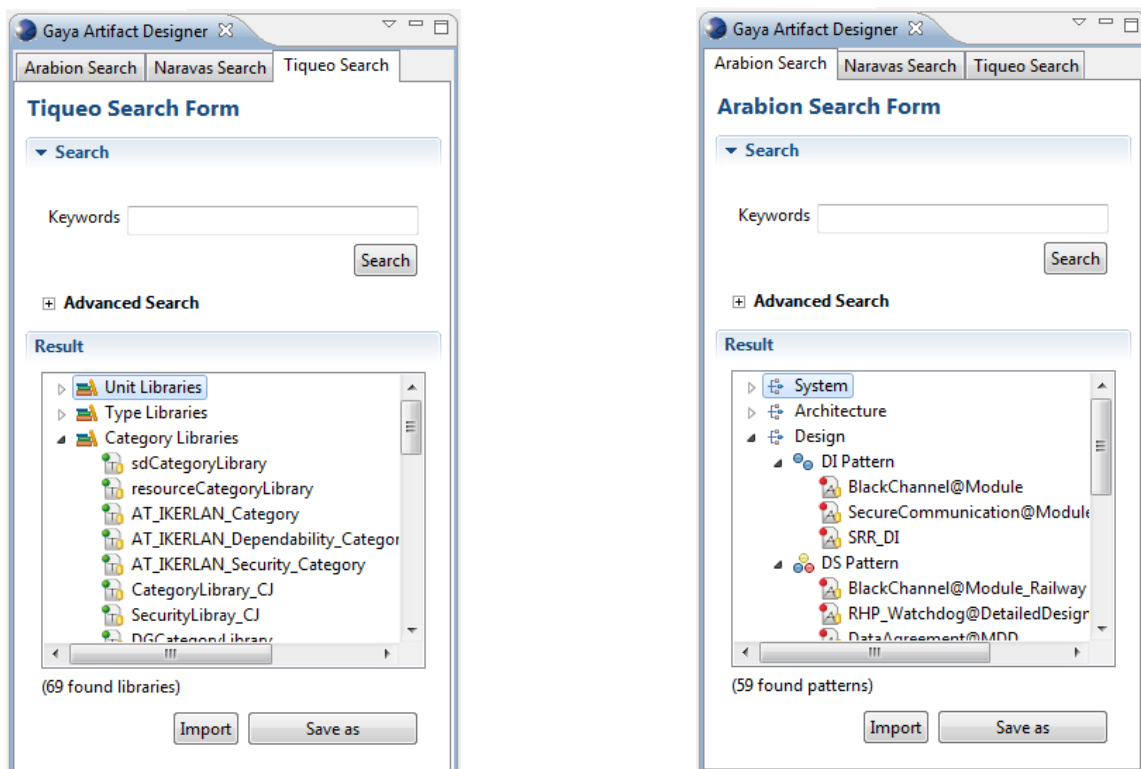


Figure 5.20: Repository Content

The following tables depict the subset of the railway pattern language for the Safe4Rail application and the subset of metrology pattern language for the Metering Gateway application, respectively.

<i>Pattern</i>	<i>Security / Dependability</i>	<i>Domain Supported</i>	<i>Development Stage</i>
SafetyCommLayer	Dependability	Domain Independent	System Concept
			System Architecture
			Software Architecture
		Railway Domain	Detailed Design
			System Concept
			System Architecture
Hypervisor	Dependability	Domain Independent	System Concept
		Railway Domain	System Architecture
		Domain Independent	System Concept
MajorityVoter	Dependability	Railway Domain	System Architecture
		Domain Independent	Software Architecture
		Detailed Design	
ReciprocalMonitoring	Dependability	Domain Independent	Software Architecture
		Railway Domain	Software Architecture
		Detailed Design	
TMR	Dependability	Domain Independent	System Concept
		Railway Domain	System Architecture
		System Concept	
SecurityCommLayer	Security	Domain Independent	System Architecture
		Railway Domain	System Architecture
		Software Architecture	
Watchdog	Security	Domain Independent	Detailed Design
		Railway Domain	System Architecture
		System Architecture	
DataAgreement	Dependability	Railway Domain	System Concept
			System Architecture
			Software Architecture
			Detailed Design

<i>Pattern</i>	<i>Security / Dependability</i>	<i>Domain Supported</i>	<i>Development Stage</i>
Secure Remote Readout	Security	Metrology Domain	Detailed Design Implementation
Wakeup Service	Security	Metrology Domain	Detailed Design
Secure Communication	Security	Domain Independent	Detailed Design
		Metrology Domain	
Secure Logger	Security	Metrology Domain	Detailed Design
Key Manager	Security	Metrology Domain	Detailed Design
RNG Test	Security	Metrology Domain	Unit Test
Smart Meter Gateway Skeleton	Security	Metrology Domain	Software Architecture

## 6 Outlook

In this chapter we highlight some experimentation for the tool suite evolution for external dissemination and exploitation. We examine four concerns:

- Traceability.
- Our tool-suite integration with the SIT properties editor tool (see the WP5).
- An implementation of the repository proposing a web service based standardized interface.
- A graphical version of the Arabion pattern editor tool.

### 6.1 Web Service Implementation of the Gaya Repository

We used the Eclipse EMF/CDO based Ecore technology to create our repository system as described in Section 3.1. In this section we highlight our experiment for an implementation of the repository proposing a web service based standardized interface for development environments. The related architecture elements are visualized in the right part of Figure 6.1.

Instead of deploying the Interaction API in Java on the client side, the idea of this architecture is to offer interoperable web services on the server side. The web service based architecture adds a tier on a server which acts as intermediate agent between the repository (offering a Java-based CDO API) and the client (consuming web services).

Developments and tests were conducted during an internship by a M.Sc. student and an implementation of a server offering web services has shown first results. The server part offers a generic web service interface, allowing to retrieve from and post models to a repository, as well as mechanisms for authentication. The implementation was realized using an OSGi compatible application server (JBoss<sup>1</sup>) as execution platform for the Interaction API and the supporting Eclipse plugins (e.g. EMF, CDO Client, Net4j) and the web service framework (i.e. Axis2/Java<sup>2</sup>) offering the services on the server. A client implementation has been realized using the Spring Framework<sup>3</sup> and its extensions for web services<sup>4</sup>.

---

<sup>1</sup><http://www.jboss.org/jbossas>

<sup>2</sup><http://axis.apache.org/axis2/java/core/>

<sup>3</sup><http://www.springsource.org/spring-framework>

<sup>4</sup><http://www.springsource.org/spring-web-services>



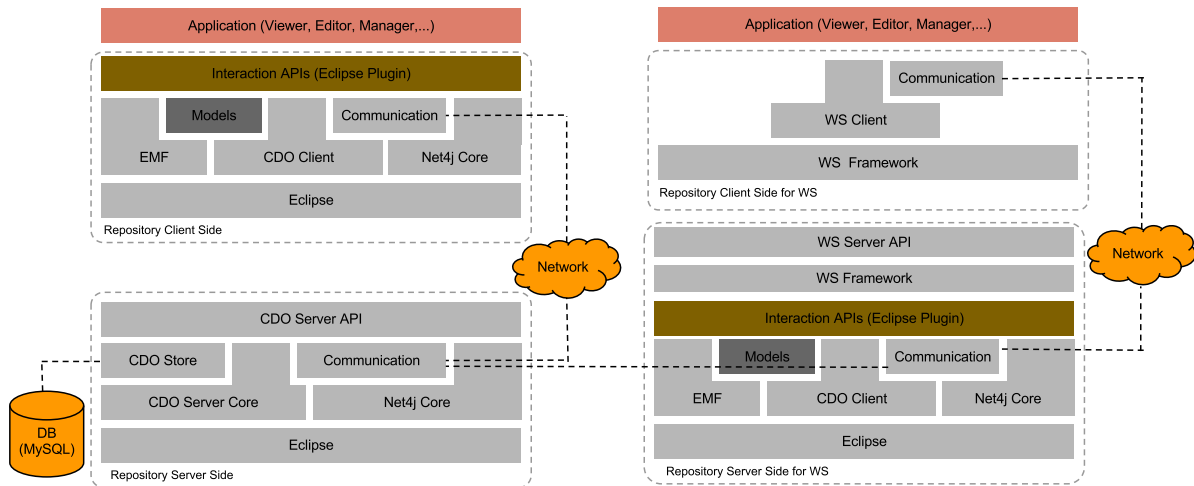


Figure 6.1: Gaya Webservice based architecture

## 6.2 Graphical Pattern Editor

Using the GMF framework<sup>5</sup>, a prototype for a graphical pattern designer is developed. As visualized in Figure 6.2, the environment is composed of three part. There is a design palette on the right, the main design view on the right and the properties view of the selected design entity on the low part.

<sup>5</sup><http://www.eclipse.org/modeling/gmp/>

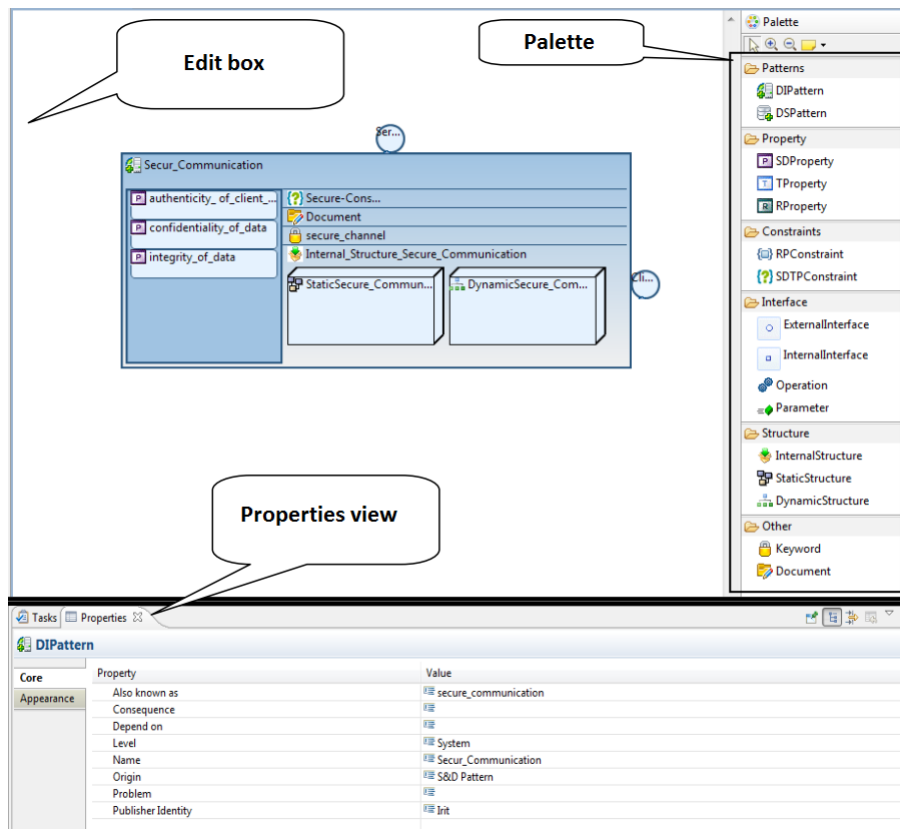


Figure 6.2: Pattern Designer with a Graphical Environment

## 7 Conclusion

---

The resulting repository system, as a data structure that stores artifacts and that allows users to publish and retrieve them, with its documentation and a number of guidelines, will facilitate 1) the population of the repository with further security and dependability patterns, and 2) the transformation of the S&D patterns into platform dependent specifications.

The pattern, properties and constraint modeling languages used in this deliverable have evolved compared to ones presented in the previous deliverables (D4.2 and D4.3). In fact, the successful evaluation by the TERESA partners, mainly for the railway domain not only resulted in a set of refinements and improvements, but it also pointed the major industrial requirements that the framework now meets. One of them is the repository storage and the other the support for interactions in the artifact and the system development lifecycles. For instance, a repository of S&D patterns allows reusing validated patterns. A pattern designer defines patterns and stores them in the repository. A system designer reuses existing patterns from the repository through the instantiation mechanisms which lead to simpler and almost seamless designs with quality improvement and cost saving. Another one is the materialization of links and references among patterns with regard to the domain, development lifecycle stage and the ones related to the pattern language itself.

As future work, new specification languages may be designed and stored with their related artifacts in the repository. For instance, components and resources are important kinds of artifacts that we can consider in our framework to serve for systematic construction of large complex systems with multiple concerns. Moreover, test, analysis and simulation artefacts may be generated for the assistance of safety development processes. As a result, specification languages, roles and compartments related to each of them can be clearly defined and applied in system development for more flexibility and efficiency. In addition, our tool suite promotes the separation of concerns during the development process by distinguishing the roles of the stakeholders. Mainly, the accessing the repository is customized regarding the development phases and the stakeholders domain and system knowledge.

In addition, we will study the integration of our tooling with other tools, mainly those used by TERESA's industrial partners. For that, we need to implement code generators able to generate a restrictive set of code complying to the domains standards. Also, we will seek new opportunities to apply the framework to other domains.

## Appendix A: Terminology and Abbreviation

- *EFF*. Extra Functional Properties
- *NFP*. Non Functional Properties
- *RCES*. Resource Constrained Embedded Systems
- *S&D*. Security and Dependability
- *PFS*. Pattern Fundamental Structure (meta-model)
- *DIPM*. Domain Independent Pattern Model
- *DSPM*. Domain Specific Pattern Model
- *SDPCM*. Security Dependability Property and Constraint Model
- *RPCM*. Resource Property and Constraint Model
- *P4SDM*. Pattern for Security and Dependability Model
- *RCPM*. Repository-Centric Process Model
- *MARM*. Modeling Artifacts Repository Model
- *eSDPCM*. Ecore Security and Dependability Property and Constraint Model
- *eRPCM*. Ecore Resource Property and Constraint Model
- *eP4SDM*. Ecore Pattern for Security and Dependability Model
- *eRCPM*. Ecore Repository-Centric Process Model
- *eMARM*. Ecore Modeling Artifacts Repository Model
- *eSDPCe*. EMF Security and Dependability Property and Constraint Editor
- *eRPCe*. EMF Resource Property and Constraint Editor
- *eP4SDe*. EMF Pattern for Security and Dependability Editor
- *eRCPe*. EMF Repository-Centric Process Editor
- *MARs*. Modeling Artifacts Repository Storage
- *MDE*. Model Driven Engineering
- *EMF*. Eclipse Modeling Framework
- *DSL*. Domain Specific Language

- *M2T*. Model to Text transformation
- *M2M*. Model to Model transformation
- *T2T*. Text to Text transformation
- *Gaya*. Repository technology
- *Naravas*. Process technology
- *Arabion*. Pattern technology
- *Tiqueo*. Property and Constraint technology

## Appendix B: Tiqueo Examples

In the following figures we show examples of the use of the Tiqueo tool to create libraries for both Security&Dependability (see Figure 7.1, Figure 7.2 and Figure 7.3) and Resource (see Figure 7.4, Figure 7.5 and Figure 7.6).

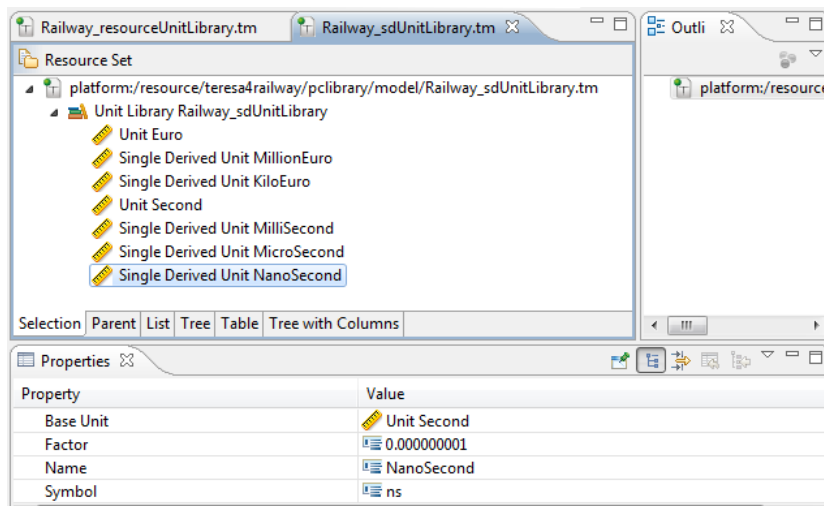


Figure 7.1: Railway S&D Unit Library

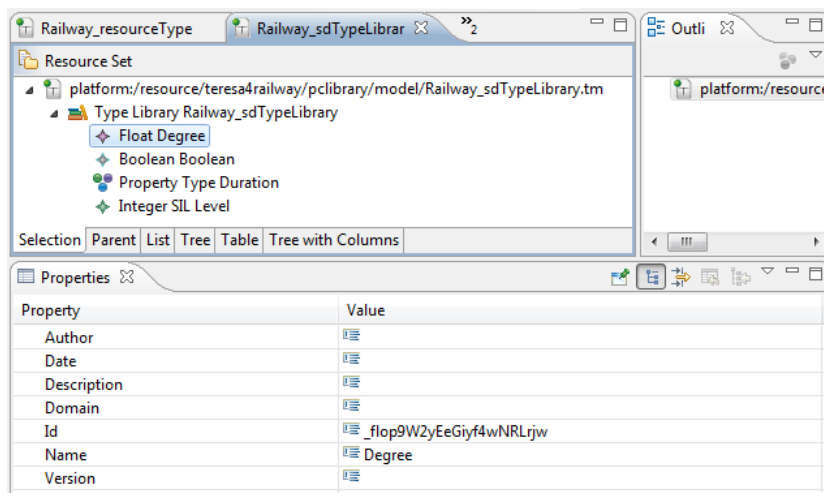


Figure 7.2: Railway S&D Type Library

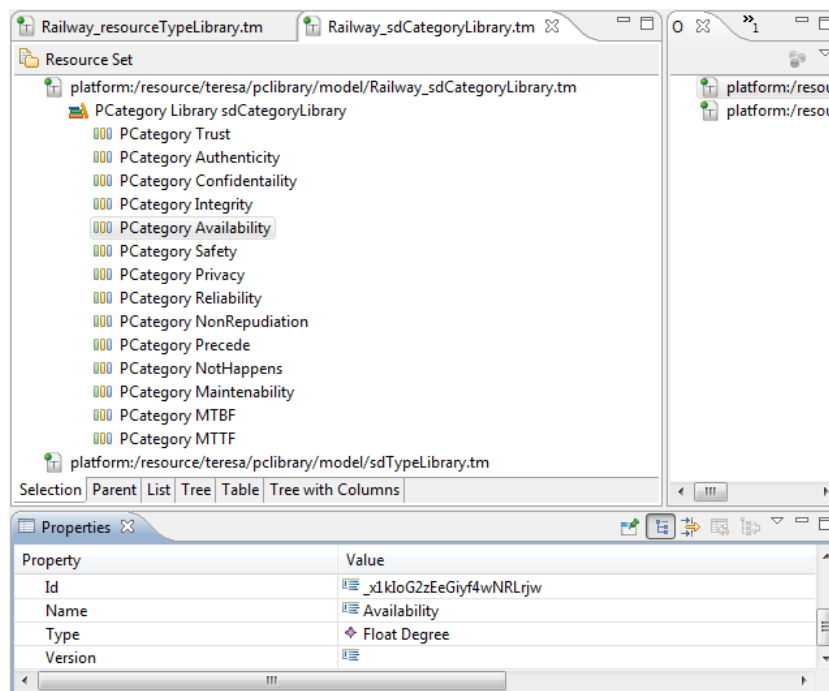


Figure 7.3: Railway S&D Property Category Library

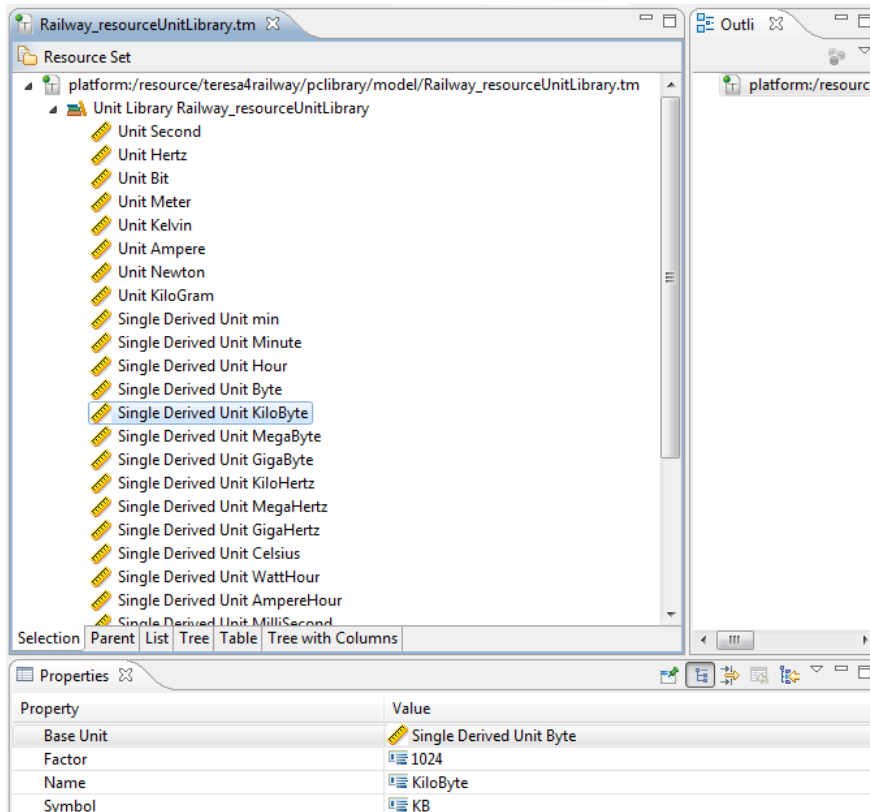


Figure 7.4: Railway Resource Unit Library



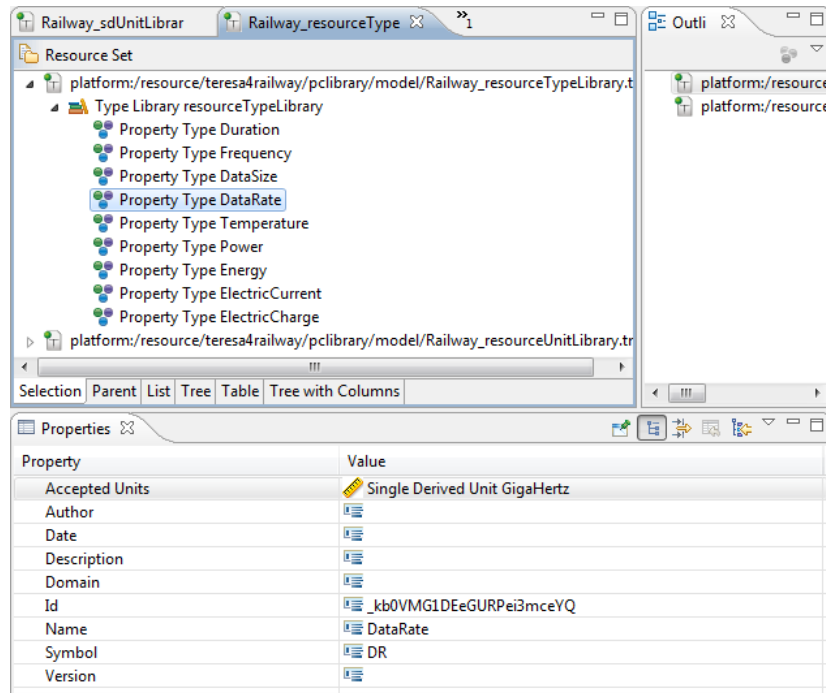


Figure 7.5: Railway Resource Type Library

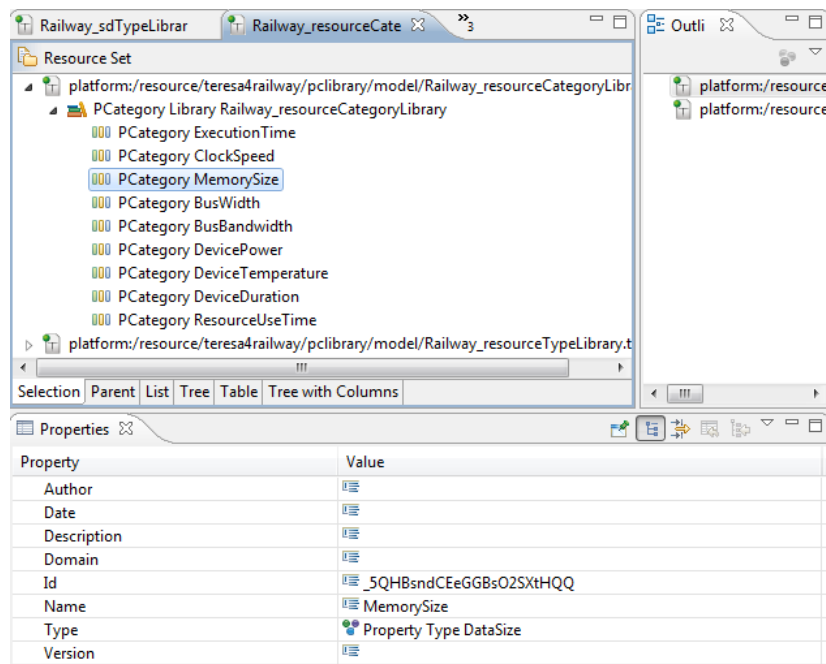


Figure 7.6: Railway Resource Property Category Library

# Appendix C: Arabion Examples

## DI Patterns

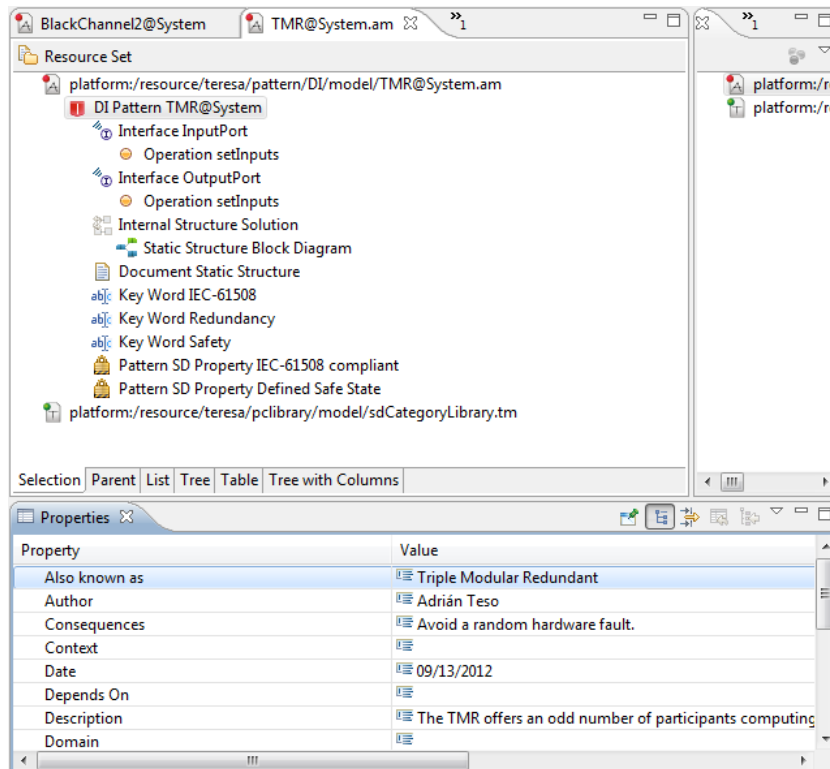


Figure 7.7: DIPattern TMR System

## DS Patterns

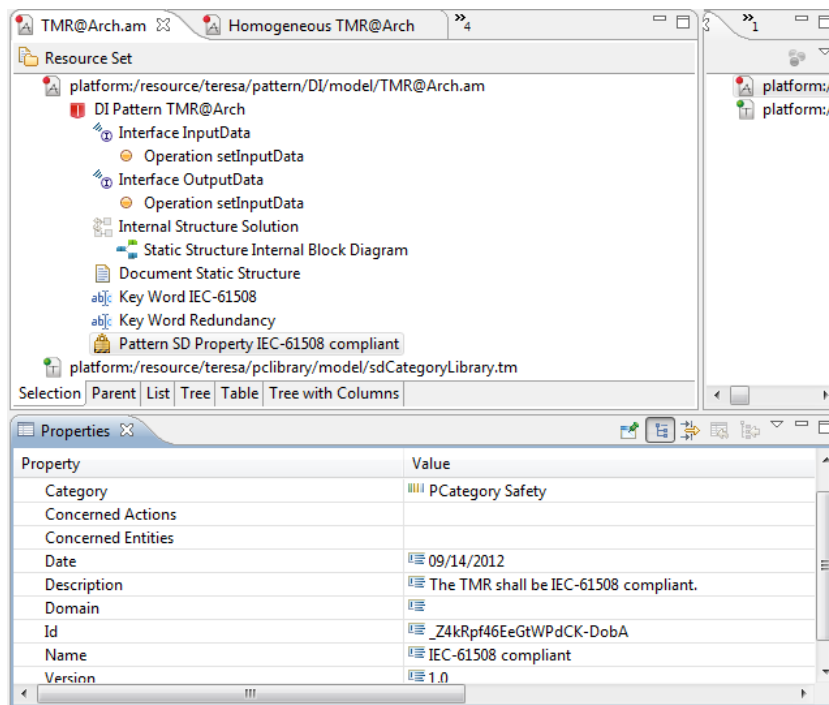


Figure 7.8: DIPattern TMR Architecture

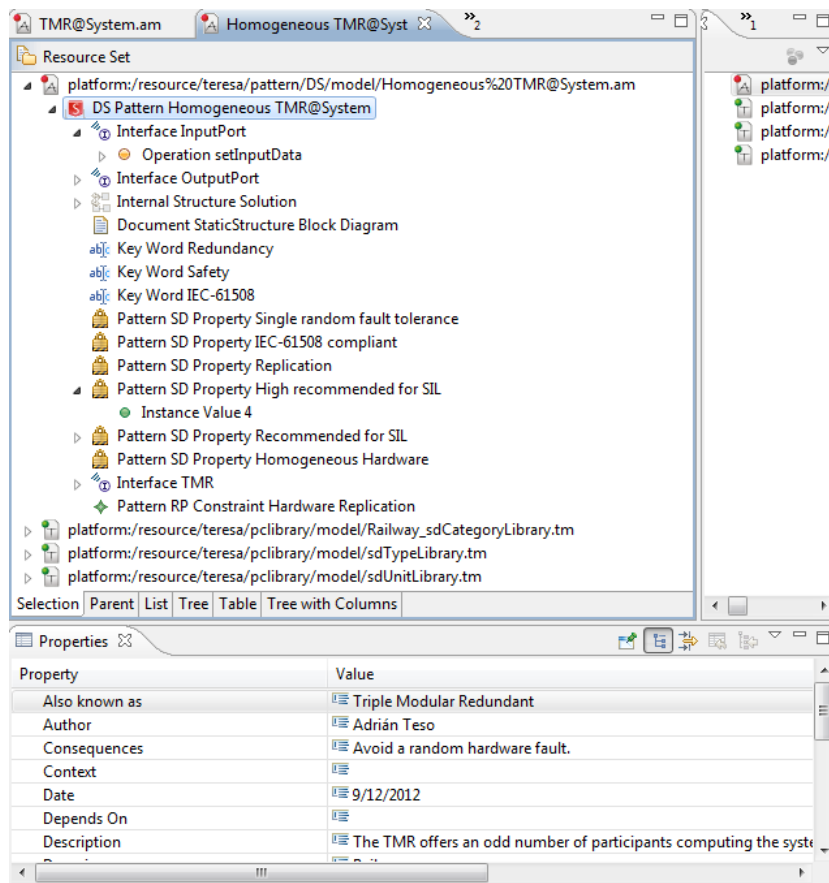


Figure 7.9: DSPattern TMR System

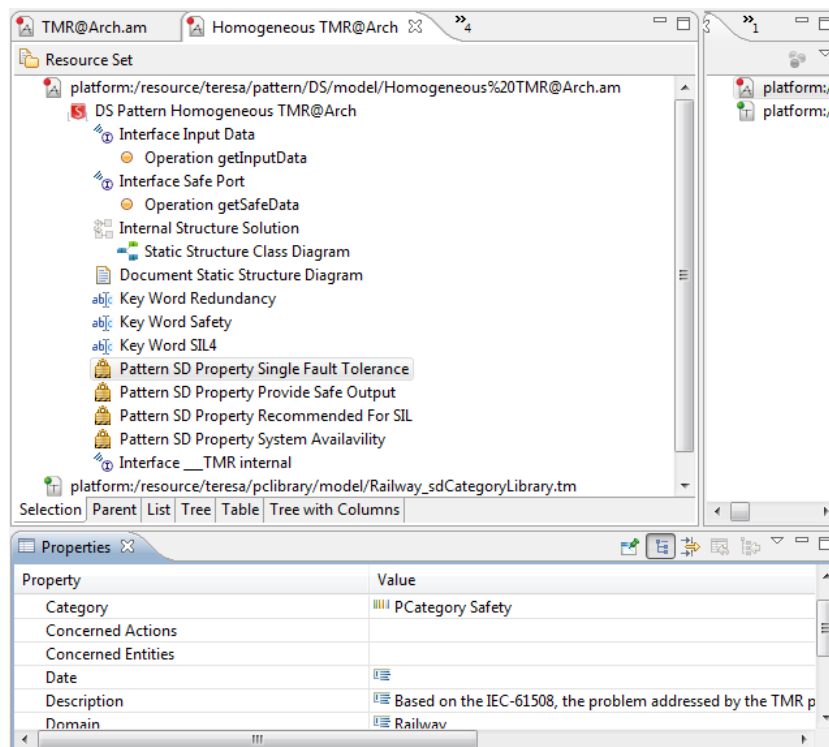


Figure 7.10: DSPattern TMR Architecture