Modeling of Secure Dependable (S&D) applications based on patterns for Resource-Constrained Embedded Systems (RCES)

Adel Ziani

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 2 Le Mirail (UT2 Le Mirail)*

Présentée et soutenue le *19 Septembre 2013* **par :**

ADEL ZIANI

Modeling of Secure and Dependable (S&D) Applications based on a Repository of Patterns for Resource-Constrained Embedded Systems (RCES)

JURY

JEAN-MICHEL BRUEL	Professeur, Université de Toulouse	Directeur de thèse
BRAHIM HAMID	Maître de Conférences, Université de Toulouse	Co-Directeur de thèse
FRANCK BARBIER	Professeur, Université de Pau	Rapporteur
FERHAT KHENDEK	Professeur, Université de Montréal - Concodia	Rapporteur
VINCENT CHAPURLAT	Professeur, Ecole des Mines d'Alès	Examinateur
MOHAMED KAÂNICHE	Directeur de Recherche, LAAS	Examinateur

École doctorale et spécialité :
MITT : Domaine STIC : Sureté de logiciel et calcul de haute performance

Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (IRIT)

Directeur(s) de Thèse :
Jean-Michel BRUEL et *Brahim HAMID*

Rapporteurs :
Franck BARBIER et *Ferhat KHENDEK*

Acknowledgements

When I was young, I always wanted to be a scientist and to invent something... I can not say that I have invented something amazing, but I can say that I have learned a lot.

I would like to thank my committee members Franck BARBIER, Ferhat KHENDEK, Mohamed KAÂNICHE, Vincent CHAPURLAT, Jean-Michel BRUEL and Brahim HAMID for their friendship and wisdom.

I am grateful to my supervisor, Prof Jean-Michel BRUEL for giving me the opportunity to be a Ph.D. student and believing in me. He helped me a lot with his detailed and precious comments. I want to thank my second assistant supervisor Dr Brahim HAMID. He inspired my interest in embedded system modeling and supported me in many ways during the becoming of this thesis. I do not want to thank him only as a supervisor, but also as an invaluable friend that has always been there for me, and has supported me many times.

Thanks also to colleagues for their characteristically sage comments, and for guidance in the ways of leadership. I am indebted to all the members of the Macao. It has been a privilege to enjoy your warm humor and insightful criticism alike.

Finally, for the love of my family, both old and new, near and far. I am grateful to my family who provided the right environment for my studies and my work. My brother and my sisters and persons who are near to my heart even though we are always far away. To my friends for making me one of the family. Thanks go also to all other people that contributed in any way to the success of this thesis. To my circle of peers I offer special thanks...

Résumé

La complexité croissante du matériel et du logiciel dans le développement des applications pour les systèmes embarqués induit de nouveaux besoins et de nouvelles contraintes en termes de fonctionnalités, de capacité de stockage, de calcul et de consommation d'énergie. Un autre défi qui s'ajoute à cette complexité est le développement des applications avec de fortes exigences de sécurité et de fiabilité (S&D) pour des systèmes embarqués contraints en ressources (RCES). Dans ce travail, nous proposons une approche d'ingénierie à base de modèles pour la spécification, le packaging et la réutilisation d'un ensemble d'artifacts pour modéliser et analyser ces systèmes, ou le "patrons" constitue l'artifact de base pour représenter des solutions S&D.

Le fondement de l'approche est un ensemble de langages de modélisation couplés à un référentiel à base modèles et de moteurs de recherche et d'instantiation vers des environnements de développement spécifiques. Ces langages de modélisation permettent de spécifier les patrons, les resources et un ensemble de modèles de propriétés. Ces derniers permettent de gouverner l'utilisation des patrons et leur analyse pour d'éventuelle réutilisation. En outre, nous proposons un processus de spécification et de génération de référentiels.

Dans le cadre de l'assistance pour le développement des applications S&D, nous avons implémenté une suite d'outils structurée autour de la plateforme Eclipse pour supporter les différentes activités autours du référentiel en passant par les activités d'analyse. Les solutions proposées ont été évaluées dans le cadre du projet TERESA à travers un cas d'étude d'une application ferroviaire.

Mots-clés: Systèmes Embarqués Contraints en Ressources, Ingénierie Dirigée par les Modèles, Référentiel basé sur des Modèles, Patrons de sécurité et de fiabilité, Suite d'outils IDM

Abstract

Non-functional requirements such as Security and Dependability (S&D) become more and more important as well as more and more difficult to achieve, particularly in embedded systems development. Such systems come with a large number of common characteristics, including real-time and temperature constraints, security and dependability as well as efficiency requirements. In particular, the development of Resource-Constrained Embedded Systems (RCES) has to address constraints regarding memory, computational processing power and/or energy consumption. In this work, we propose a modeling environment which associates model-driven paradigms and a model-based repository, to support the design and the packaging of S&D patterns, resource models and their property models.

The approach is based on a set of modeling languages coupled with a model-repository, search and instantiation engines towards specific development environments. These modeling languages allow to specify patterns, resources and a set of property models. These property models will allow to govern the use of patterns and their analysis for reuse. In addition, we propose a specification and generation process of repositories.

As part of the assistance for the development of S&D applications, we have implemented a tool-chain based on the Eclipse platform to support the different activities around the repository, including the analysis activities. The proposed solutions were evaluated in the TERESA project through a case study from the railway domain.

Keywords: Resource-Constrained Embedded Systems, Model-Driven Engineering, Model-based repository, Security and Dependability patterns, MDE tool-chain

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Recent times have seen a paradigm shift in terms of design by combining multiple software engineering paradigms, namely, Model-Driven Engineering (MDE) [10, 64] and Component Based Software Engineering (CBSE) [14, 15, 68]. Such a paradigm shift is changing the way systems are developed nowadays, reducing development time significantly. Embedded systems [88] are a case where a range of products for assorted domains such as energy, transportation, automotive, and so on are conceived as a family. However, most of the work so far has been focused on the functional parts.

Embedded systems [35, 88] are not conventional software which can be built using usual paradigms. In particular, the development of Resource-Constrained Embedded Systems (RCES) addresses constraints regarding memory, computational processing power and/or limited energy. To cope with the growing complexity of embedded system design, several development approaches have been proposed. The most popular are those using models as main artifacts to be constructed and maintained. In these processes, software development consists of model transformations.

Non-functional requirements such as Security and Dependability (S&D) [59] become more important as well as more difficult to achieve. The integration of S&D features requires the availability of both application domain specific knowledge and S&D expertise at the same time. Currently, the integration of S&D mechanisms is still new in many domains (i.e. smart metering or home control), hence embedded system developers usually have limited S&D expertise. In fact, capturing and providing this expertise by the way of *S&D patterns* can support embedded system

development.

Unfortunately, most of S&D patterns are expressed as informal indications on how to solve some security problems, using template like traditional patterns. These patterns do not include sufficient semantic descriptions, including those of security and dependability concepts, for automated processing within a tool-supported development and to extend their use. Furthermore, due to manual pattern implementation, the problem of incorrect implementation (the most important source of security problems) remains unsolved. For that, model driven software engineering can provide a solid basis for formulating design patterns that can incorporate security and dependability aspects and for offering these patterns at several layers of abstraction. We leverage on this idea to propose a new framework for the specification and the management of a set of modeling artifacts, including S&D patterns, resource models and property models, intended for systems with stringent S&D requirements. Reaching this target requires to get a common representation of such a modeling environment for several domains and the ability to customize them for a specific domain.

The industrial context conducting our work is how to take into account several constraints, mainly those related to security and dependability, that are not satisfied by the well-know and the widely used technology for building applications for Resource-Constrained Embedded Systems. These requirements introduce conflicts on the three main factors that determine the cost of ownership of applications: cost of the production, cost of engineering and cost of maintenance. In other words, systems with high dependability requirements for which the security level must be demonstrated and certified use almost exclusively technical solutions strongly oriented by the application domains. Applications based on these solutions are by definition dedicated, hardly portable between different execution platforms and require specific engineering processes. These specificities greatly increase the cost of the development in the phases of their lifecycle.

1.2 Problem Statement

Based on the previous section, we identify our general research problem coming from the embedded systems practice as:

The need to address the problem of defining a tool-supported process and deriving the necessary tools for the development of Resource-

Constrained Embedded System (RCES) applications with strong constraints on Security and Dependability (S&D).

This leads us to deal with the problem of enforcing S&D in RCES with Model-Driven Engineering (MDE). MDE provides a useful contribution to the design of RCES applications since it bridges the gap between design issues and implementation concerns. In addition, MDE can potentially maintain the separation of concerns between application and S&D, by ensuring that S&D designs can be reused at different development levels by application designers and developers.

Significant research is being carried out concerning MDE for embedded systems, at the level of system architecture, design techniques, testing, validation, proof of correctness, modeling, software reliability, operating systems and parallel and real-time processing. More research is needed on the use of MDE to enforce the integration of S&D requirements into the engineering process and to support the reuse of S&D mechanisms.

1.3 Research Goals

Taking into account the previous discussion, we specify our research problem as an overall research goal of this thesis:

Define, demonstrate and validate an engineering discipline for S&D that is adapted to Resource-Constrained Embedded Systems.

Special emphasis will be devoted to promote the particularly challenging task of efficiently integrating S&D solutions within the restricted available design space for RCES. Furthermore, one important focus is on the potential benefits of the combination of Model-Driven Engineering with a pattern-based representation of security and dependability solutions. Decomposing the overall research goal, we formulate three research goals that we address in this thesis.

Research goal 1.

Develop languages for modeling artifacts and tool support for modeling and analysis of RCES application based on S&D patterns. ***RG1***

Research goal 2.

Develop a language for the specification and generation of a model-based repository of modeling artifacts. ***RG2***

Research goal 3.

Study the applicability of the proposed framework by modeling and analyzing a case study from industry. ***RG3***

1.4 Contributions

The proposed approach is to use a model-based repository of S&D patterns and resource models augmented with property models. Some of the topics that we seek to include in this work are related to the development of models and tools to support the inclusion of security and dependability (S&D) issues into the RCES engineering process. Here, we map the contributions of this thesis to the goals formulated earlier.

RG1 is addressed with the following contributions:

1. **Artifacts modeling:** we propose a modeling framework to specify properties, resources and patterns. (Chapter 4)

2. **Model transformation:** we propose a model-to-model and a model-to-text transformation to estimate the resource consumption of S&D applications within a specific platform. (Chapter 4)

3. **Tooling:** we propose an MDE tool-chain as a set of integrated tools to support the specification the analysis of S&D pattern-based applications for RCES. (Chapter 6)

RG2 is addressed with the following contributions:

1. **Repository modeling:** we propose a modeling framework to specify a model-based repository independently from end-development applications and execution platforms. (Chapter 5)

2. **Repository tooling:** we propose a tool to support the specification of model-based repositories as well as accessing and managing these repositories. (Chapter 6)

RG3 is addressed with the following contribution:

1. **Validation:** we apply in practice to a resource-constrained embedded system (RCES) in the context of the TERESA project [28]. (Chapter 7)

1.5 Publications

This section presents published papers related to the thesis. The publications are divided into two categories: (i) papers that are fundamental for the thesis contributions; and (ii) papers that are related to the thesis.

Fundamental publications

- **Paper A.** [86] *A Model-based Repository of Security and Dependability Patterns for Trusted RCES* **Adel Ziani**, Brahim Hamid, Jean-Michel Bruel. *14th IEEE International Conference on Information Reuse and Integration*, San Francisco, CA. USA, **IEEE Computer Society**, August 2013.
  Summary: In this paper, we target the development of a model-based repository of S&D patterns that follows the MDE paradigm. Our framework is based on metamodeling techniques that allow to specify the S&D patterns at different levels of abstraction and an operational architecture of the repository. Furthermore, we walk through a prototype with EMF editors and a CDO-based repository supporting the approach. Currently the tool suite named semcomdt is provided as Eclipse plugins. The approach presented here has been evaluated in the context of the TERESA project for a repository of S&D patterns and property models targeting RCES applications. First evidences indicate that users are satisfied with the notion of "model-based repository of S&D patterns". The approach paves the way to let users define their own road-maps upon the PBSE methodology. First evaluations are encouraging with 85% of the subjects being able to complete the tasks. However, they also point out two main challenges: pattern integration and automatic search for appropriate patterns.

- **Paper B.** [33] *Towards Tool Support for Pattern-Based Secure and Dependable Systems Development.* Brahim Hamid, **Adel Ziani**, Jacob Geisel. *Academics tooling with Eclipse (A joint ECMFA/ECSA/ECOOP workshop)*, Montpellier, France, **ACM DL**, July 2013.

Summary: In this paper, we present the SEMCO MDE Tool Suite development status conducted in the context of the FP7 TERESA project aiming to support the automation of building, storing and processing reusable artifacts (S&D patterns and property models). This tool promotes the PBSE methodology in the domain of assistance to the trusted embedded system engineering. The approach presented here has been evaluated in the context of the TERESA project for a repository of S&D patterns and property models. For instance, a pattern designer defines patterns and store them in the repository. A system designer reuses existing patterns from the repository through instantiation mechanisms which leads to simpler and seamless designs with higher quality and costs savings.

- **Paper C.** [29] *Model-Driven Engineering for Trusted Embedded Systems based on Security and Dependability Patterns.* Brahim Hamid, Jacob Geisel, **Adel Ziani**, Jean-Michel Bruel, Jon Perez. *System Design Languages Forum*, Montreal, Canada, **Springer, LNCS**, p. 73-91, June 2013.
  Summary: In this paper, we propose a methodology and an MDE tool-chain to support the specification and the packaging of a set of S&D patterns, in order to assist the developers of trusted applications for resource-constrained embedded systems.

- **Paper D.** [87] *Towards a Unified Meta-model for Resources-Constrained Embedded Systems.* **Adel Ziani**, Brahim Hamid, Salvador Trujillo. Dans : *Euromicro conference on Software Engineering and Advanced Applications, Oulu, Finland, 30/08/2011-02/09/2011* **IEEE Computer Society**, p. 485-492, September 2011.
  Summary: In this paper, we introduce a model-driven approach for modeling of non-functional properties in the context of resource-constrained embedded systems. The RCES metamodel serves primarily to capture the basic concepts constituting an embedded systems (resources, services and properties). Based on these building blocks, the metamodel extends the concept of property to express non-functional properties. Subsequently, the RCES properties model is defined according to the metamodel comprising a set of properties types, units of measure and predefined properties. To demonstrate the use of our metamodel and model to define properties specific to a platform, a use case illustrates the mechanisms available to do it. The benefits of this work is twofold: first, RCES model serves for analysing the modeling of embedded

systems. Second, the properties provided by such models will be used as basis for constructing constraints to build RCES applications.

Publications related to the thesis

- [30] *Safety Lifecycle Development Process Modeling for Embedded Systems - Example of Railway Domain* . Brahim Hamid, Jacob Geisel, **Adel Ziani**, David Gonzalez. Dans : *Software Engineering for Resilient Systems (SERENE 2012)*, Pisa, Italy, 27/09/2012-28/09/2012, Vol. 7527, **Springer, LNCS**, p. 63-75, September 2012.

- [85] *A Model-Driven Engineering Framework for Fault Tolerance in Dependable Embedded Systems Design.* **Adel Ziani**, Brahim Hamid, Jean-Michel Bruel. Dans : *Euromicro conference on Software Engineering and Advanced Applications, Cesme, Izmir, Turkey, 05/09/2012-08/09/2012*, **IEEE**, p. 166-169, September 2012.

- [32] *An Environment for Design Software and Hardware Aspects of Clock Synchronization and Communication in DRTES.* Brahim Hamid, **Adel Ziani**. Dans : *IEEE International Conference on Embedded and Ubiquitous Computing (EUC 2010), Hong Kong SAR, China, 11/12/2010-13/12/2010* **IEEE Computer Society - Conference Publishing Services**, p. 60-67, December 2010.

- [84] *Clock Synchronization Modeling in DRTES.* **Adel Ziani**, Brahim Hamid. Dans : *Handson Platforms and tools for model-based engineering of Embedded Systems (workshop at ECMFA 2010)(HoPES 2010)*, Paris, 15/06/2010-16/06/2010, **CEA LIST**, p. 51-56, 2010.

1.6 Thesis Outline

The outline of the dissertation is as follows. Chapter 2 presents the context of our work. Chapter 3 is dedicated to relating the contributions in this thesis to relevant research. Chapter 4 presents the modeling languages and available tools for analysis to help the development of applications based on S&D patterns. Chapter 5 presents the specification and the generation of repositories for the packaging of models and S&D patterns. Chapter 6 presents the MDE tool-suite to support the design process,

the packaging and the analysis of S&D applications around a repository and methods introduced to promote its use. An illustration of the use of our tool-chain and the benefits of our proposed solutions within an industrial case study *Safe4Rail*, outcome of the TERESA project, is presented in Chapter 7. Finally, Chapter 8 concludes the dissertation and proposes some perspectives on the future works.

Chapter 2

Context

2.1 Introduction

The conception and design of RCES is an inherently complex endeavor. In particular, non-functional requirements such as Security and Dependability (S&D) are exacerbating this complexity. MDE is a promising approach for the design of trusted systems, since it bridges the gap between design issues and implementation concerns. MDE has the potential to greatly ease recurring activities of S&D experts. MDE supports the designer to resolve in a separate way non-functional requirements such as security and/or dependability issues at a greater abstraction level.

In this chapter we present the context of our work, including a set of concepts, definitions and an introduction to the case study that might prove useful in understanding our approach.

2.2 Resource-Constrained Embedded Systems (RCES)

Embedded systems come with a large number of common characteristics, including real-time, temperature and energy constraints, dependability as well as efficiency requirements [35]. Especially, Resource-Constrained Embedded Systems (RCES) refer to systems which have memory, computational processing and/or energy consumption constraints. They are commonly found in many application sectors such as automotive, aerospace and home control. They are in many types of device like sensors, automotive electronic control units, intelligent switches and home appliances. In addition, they are heterogeneous, e.g. standalone systems, peripheral subsystems

and main computing systems. Embedded resources of RCES, e.g. memory, tasks and buffers are generally statically determined. Therefore, the generation of RCES involves specific software building processes. These processes are often error-prone because not fully automated, even if some level of automatic code generation or even Model-Driven Engineering support is used.

2.2.1 The TERESA project

In the context of the TERESA project [74], Resource-Constrained Embedded Systems are characterized as follows:

- They belong to different application sectors.
- Computing resources are mostly statically determined and allocated through a process consisting of a configuration phase and a build phase.
- They are generally high integrity systems with strong assurance requirements, ranging from very strong levels involving certification (e.g. DO178 and IEC-61508 for safety-relevant embedded systems development) to lighter levels based on industry practices.
- They therefore use advanced engineering disciplines.

TERESA (Trusted Computing Engineering for Resource-Constrained Embedded Systems Applications) planned to define, demonstrate and validate an engineering discipline for trust that is adapted to resource-constrained embedded systems. In TERESA, trust is defined as the degree with which security and dependability requirements are met. TERESA has the following objectives: (1) Provide guidelines for the specification of sector specific RCES trusted computing engineering. Software process engineers in a given sector can then use the guidelines to define a trusted computing engineering process that is integrated with the software engineering process used in their RCES sector. (2) Define a trusted computing engineering approach that is suited to the following sectors: Automotive, Home Control, Railway and Metering.

2.2.2 Resource-Aware System Engineering

Embedded systems can be defined as information processing systems embedding hardware and software into enclosing products to fulfill a specific function, often

under real-time computing constraints [47]. The resources that such systems use (CPU, memory, energy, etc.) have a fixed nature, they are limited in capacity, expensive and usually not extensible during the lifetime of the system [79]. In contrast to this fixed nature, software can be subject to change. To ensure that the combination of software fits on the selected hardware platform, we need to be able to estimate the resources consumed by the application software. Prediction methods for resource usage should be available at early stages of design to help designer to prevent resource conflicts at run-time.

2.3 Security and Dependability

In this section, we present the TERESA S&D conceptual model. The goal of this model is to illustrate the addressed problems and to have a common understanding of all the concepts used in this thesis. In the following text, all concepts are typeset in bold letters at the moment of their definition. The conceptual model is composed of three models presented in the following sections.

A system is a combination of interacting elements organized to achieve one more stated purposes [12]. In TERESA, we make use of this model to comprehend all the factors that may lead to an error of an embedded system and hence to determine the trust level of a system. Basically, the trust level of a system decreases if a fault could damage the system.

The Fault Propagation Model. Typically, a fault can have a direct influence on S&D. Generally, the AVI (Attack, Vulnerability, and Intrusion) fault model is used to demonstrate the fault $->$ error $->$ failure paradigm (see the Figure 2.1) [57]. Figure 2.1 explains how a fault could be propagated into the system and how it could lead to a failure.

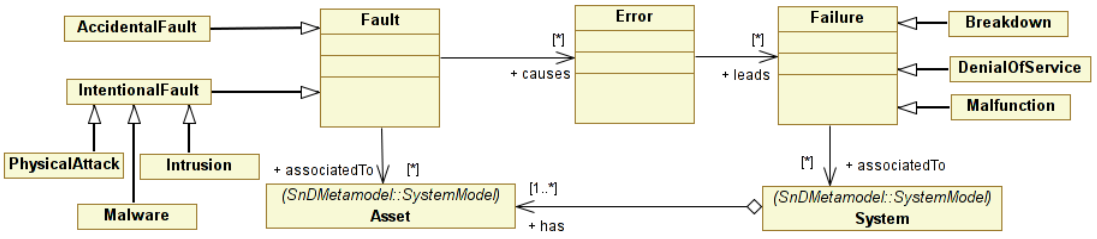

Figure 2.1: Fault Propagation Model

As shown in Figure 2.1, a **fault** is the presumed or hypothesized cause of an **error**. Faults may be classified according to several criterion. A fault is related to an asset of the system. An **asset** could be hardware, software, people, etc...Here, we classify faults as **accidental** or **intentional**. Accidental faults can arise during either the system design and development process or during operation (through a violation of an operating or maintenance procedure).

Intentional faults fall into three classes: malware, physical attacks and intrusions. Typically an attack exploits a fault or vulnerability of an asset to make an intrusion. An attack may use physical means to cause faults, such as power fluctuations, radiation or wire-tapping.

An error is detected if its presence in the system triggers an error message or error signal that originates from within the system. The possibility to detect errors contributes to improved security and trust in the system. Errors that are present but not detected are called latent errors. In [7], Avizienis defines an error as the part of the system state that may lead to a failure. A failure leads to the inability of the system or some of its parts to meet their specifications (functional and non-functional requirements). Possible failures could be: breakdown, denial of service and malfunction.

The Trust Model. In order to limit the probability of occurrence as well as the impact of a fault, TERESA uses a Trust Model, depicted in Figure 2.2, to visualize the dependencies of a RCES's trust that is built on a number of factors.

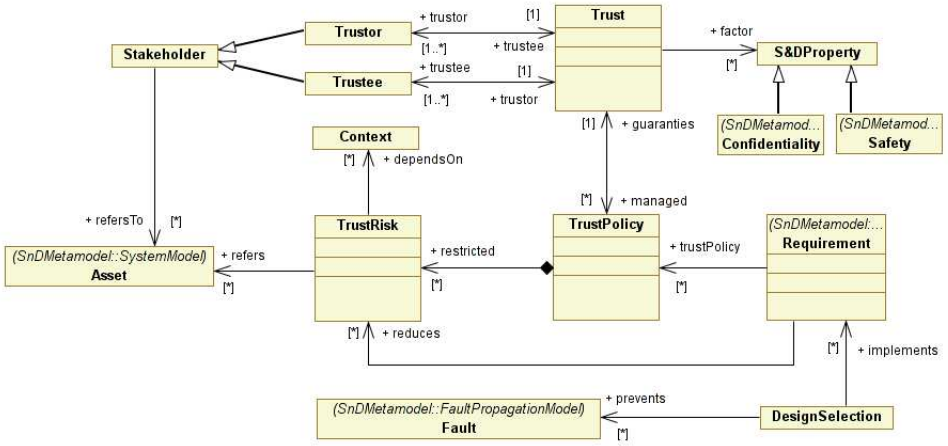

Figure 2.2: Trust Model

As depicted in Figure 2.2, a trustor can trust a trustee if, and only if, at least one

factor of S&D is guaranteed (these factors are called S&D Property in Figure 2.2). For this reason, during the development process, requirement identification is a main issue [40]. In order to fulfill the requirements, it is possible to define policies which enforce the trust level. Then, during the system implementation, design selection will implement the requirements and reduces the risk of fault.

The S&D Properties Model. While the Trust Model in Figure 2.2 figures out that trust is related to several factors or dimensions [4], the model in Figure 2.3 summarizes possible criteria in S&D. It shows, for instance, that a user could trust a system because it is confidentiality-aware and will protect his personal data. Of course, it is needed that design selections implement mechanisms which ensure this quality of service (e.g. confidentiality, authenticity, replication, etc).

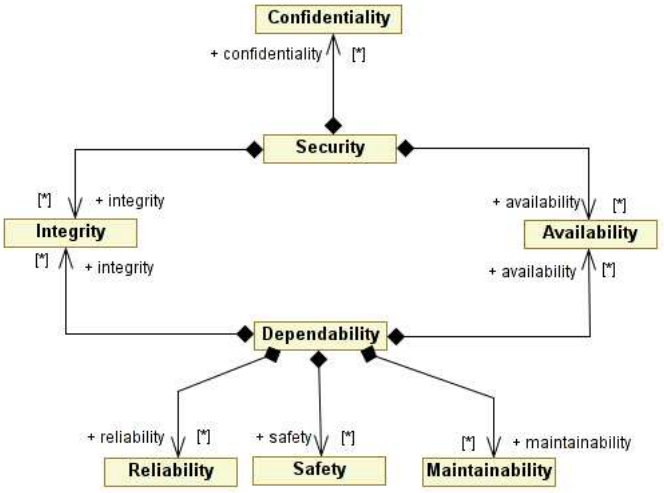

Figure 2.3: S&D Properties

2.4 Model-Based Engineering (MBE)

Models are used to denote abstract representation of computing systems. Especially, we need models to represent software architecture and software platforms to test, to simulate and to validate the proposed solutions. Model-Based Engineering (MBE) based solutions seem very promising to meet the needs of S&D RCES applications development. The idea promoted by MBE is to use models at different levels of abstraction for developing systems. In other words, models provide

input and output at all stages of system development until the final system itself is generated.

2.4.1 Model Driven Engineering (MDE)

The concept of model is becoming a major paradigm in software engineering. Its use represents a significant advance in terms of level of abstraction, continuity, generality, scalability, etc. Model-Driven Engineering (MDE) is a form of generative engineering [64], in which all or a part of an application is generated from models. MDE is a promising approach since it offers tools to deal with the development of complex systems improving their quality and reducing their development cycles. The development is based on model approaches, metamodeling, model transformation, development process and execution platforms. The advantage of having an MDE process is that it clearly define each step to be taken, forcing the developers to follow a defined methodology. MDE allows to increase software quality and to reduce the software systems development life cycle. That is, a same model is used for all businesses and, thus, the consistency is ensured by construction. Moreover, from a model it is possible to automatize steps by model refinements and generate code for all or parts of the application.

MDE provides a useful contribution for the design of trusted systems, since it bridges the gap between design issues and implementation concerns. It helps the designer to specify in a separate way non-functional requirements such as security and/or dependability needs at a higher level of abstraction. This allows implementation independent validation of models, generally considered as an important assurance step.

The development process cycles are mainly iterative, resulting in different levels of refining models of analysis and design. There are implementation platforms that address these issues in specific context (e.g. the MDA standard [10]), but in many other contexts, the links between models refined or processed to solve references (to non-existent elements, elements not referenced, created elements,...) are still solved in ad-hoc manner, without adequate support from generic technologies. The required solutions involve generally more reliable process, which essentially ensure consistency and traceability of produced models. We are still waiting for widely-applicable technologies that solve these problems in RCES environment.

A model transformation specifies mechanisms to automatically create target models from source models. The Object Management Group (OMG) defines a model

transformation as: *the process of converting a model into another model of the same system* [46]. Similarly, [43] defines model transformation as the *automatic generation of a target model from a source model, according to a transformation description.*

The Meta-Object Facility (MOF) [52] is a standard defined by the OMG to describe modeling languages such as the Unified Modeling Language (UML) [55]. Query View Transformation (QVT) [50], based on the Object Constraint Language (OCL) [56], is an OMG standard to specify model transformations in a formal way, between metamodels conforming to MOF.

2.4.2 Domain Specific Modeling Language (DSML)

A language is defined by an abstract syntax, a concrete syntax and the description of semantics [22, 34, 42]. The abstract syntax defines the concepts and their relationships which is often expressed by a metamodel. On the one hand, the concrete syntax defines the appearance of the language. In this way, a grammar or regular expressions is most of the time used to design this one. On the other hand, semantics define the sense and meaning of the structure by defining sets of rules.

Domain Specific Modeling (DSM) in software engineering is used as a methodology using models as first class citizens to specify applications within a particular domain. The purpose of DSM is to raise the level of abstraction by only using the concepts of the domain and hiding low level implementation details [26]. A Domain Specific Language (DSL) typically defines concepts and rules of the domain using a metamodel for the abstract syntax, and a concrete syntax (graphical or textual). DSLs allow to specify systems in a domain-friendly manner. As we shall see, processes in Domain Specific Modeling reuse a lot of practices from Model-Driven Engineering, for instance, metamodeling and transformation techniques.

2.5 Security and Dependability Patterns

A pattern deals with a specific, recurring problem in the design or implementation of a software system. It captures expertise in the form of reusable architecture design themes and styles, which can be reused even when algorithms, components implementations, or frameworks cannot. Today, design patterns are considered as fundamental technique to build software by capitalizing knowledge to solve occurring problems (in many specific domains). The design patterns for software building is derived from the Alexander's notion of patterns for Architecture [5] and a definition

has been proposed by Buschmann in [13]: *A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution.* That is, patterns support the construction of software with defined functional and non-functional properties. Design patterns are medium-scale patterns comparing to architectural patterns, but they are at a higher level than the programming language. The application of a design pattern has no effect on the fundamental structure of a software system, but may have a strong influence on the architecture of a subsystem (components).

In this thesis we focus on the security aspects. Security is an important non-functional requirement in software. In 1997, Yoder and Barcalow [81] were the first to work on security pattern documentation. A security pattern is a well-understood solution to a recurring information security problem. The typical structure of a security pattern is as follow [65]:

- Name: name of the security pattern.
- Context: the security context describes the conditions where the security problem occurs.
- Problem: description of the problem.
- Solution: techniques, structures and mechanisms to solve the problem.
- Forces: define the types of trade-offs that must be considered in the presence of conflicts they might create.
- Related patterns.

The concept of security pattern as a well-understood solution to a recurring information security problem was introduced to support the system engineer in selecting appropriate security or dependability solutions. However, most security patterns are expressed in a textual form, as informal indications on how to solve some (usually organizational) security problems. Some of them use more precise representations based on UML diagrams, but these patterns do not include sufficient semantic descriptions in order to automate their processing and to extend their use. Furthermore, there is no guarantee of the correct application of a pattern because the description does not consider the effects of interactions, adaptation and combination. This makes them not appropriate for automated processing within a tool-supported development process. Finally, because this type of patterns is not designed to be

integrated into the user systems but to be implemented manually, the problem of incorrect implementation (the most important source of security problems) remains unsolved.

In software engineering, patterns are considered as an efficient tool to reuse specific knowledge. For security and dependability we can encapsulate knowledge in the design of such systems through the definition of specific design patterns. For instance, communication patterns are well suited to be used in embedded real-time systems. Then, the implementation may be achieved using UML profiles.

The recurring appearance and the use of some of these patterns led to building a catalog of patterns to encode the best practices of each field. An interesting challenge is to address the problem of automation of the application of these design solutions. The difficulty is that the design solutions proposed by design patterns even differ in their details, while remaining similar in their principles. In addition, a design pattern is by construction not "complete", since it is devoted to allow many uses according to small variations around the same field.

Patterns for dependability. An hybrid set of patterns to be used in the development of fault-tolerant software applications is described in [18]. These patterns are based on classical fault tolerant strategies such as *N*-Version programming and recovery block, consensus, voting, etc. In addition, the hybrid pattern structure can be constructed through recursive combination of N-Version programming. This work addressed also the power of the technique through the support of the advanced software voting techniques. [76] proposed a framework for the development of dependable software systems based on a pattern approach. They reused proven fault tolerance techniques in form of Fault Tolerance Patterns. The pattern specification consists of a service-based architectural design and deployment restrictions in form of UML deployment diagrams for the different architectural services. The work is illustrated with an application to guide the self-repair of the system after the detection of a node crash.

Patterns for security. A collection of patterns to be used when dealing with application security is studied in [81]. The proposed catalog includes secure access layer, single access point, check point, etc. The work of [24] reports an empirical experience, about adopting and eliciting S&D patterns in the Air Traffic Management (ATM) domain, and showing the power of using patterns as a guidance to structure the analysis of operational aspects when they are used at the design stage. A survey

of approaches to security patterns is proposed in [82].

The following figure (Figure 2.4) depicts a set of most used patterns to secure Internet applications:

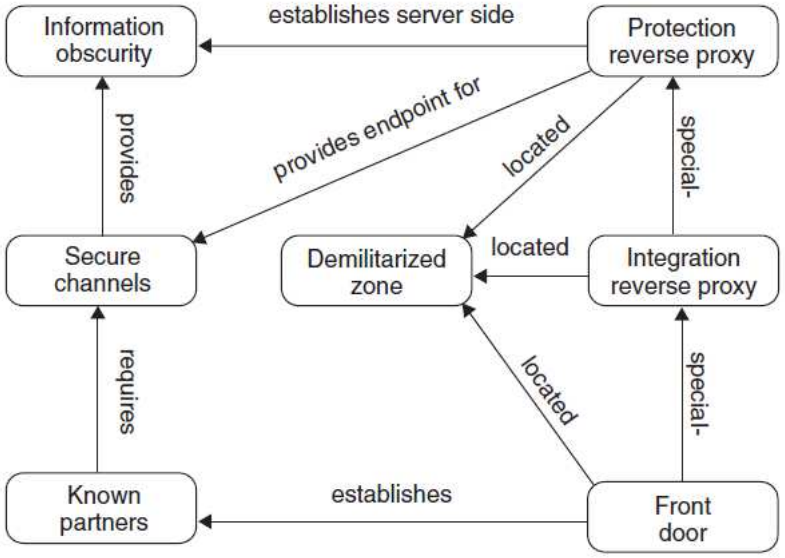

Figure 2.4: Example of some patterns to secure Internet applications

2.6 Eclipse Modeling Framework Tools

There are several DSM environments, one of them being the open-source Eclipse Modeling Framework (EMF) [67]. EMF provides an implementation of EMOF (Essential MOF), a subset of MOF, called Ecore[1]. EMF offers a set of tools to specify metamodels in Ecore and to generate other representations of them, for instance Java.

In our context, we use the Eclipse Modeling Framework (EMF). Note, however, that our vision is not limited to the EMF platform. Here, we outline the different Eclipse tools used in the development of the DSLs to support the modeling of the S&D artifacts, the repository and its APIs. Among the tools used here are cited:

- *Eclipse* is an open-source software project providing a highly integrated tool platform. The applications in Eclipse are implemented in Java and target many operating systems including Windows, Mac OSX, and Linux [67].

[1]Ecore is a meta-metamodel

- *EMF* is a modeling framework and code generation facility for building applications based on a structured data model. In addition, EMF provides the foundation for interoperability with other EMF-based tools and applications [67].
- *QVT-O* (QVT Operational) [2] allows the implementation of model-to-model transformation in Eclipse.
- *Acceleo* [49] allows the implementation of model-to-text transformation in Eclipse.
- *CDO* (Connected Data Objects) Model Repository is a 3-tier distributed shared model framework for EMF models and metamodels [1].
- *RCP plugin* allows developers to use the Eclipse platform to create flexible and extensible desktop applications upon a plug-in architecture [20, 45].

2.7 Development Environment: SEMCO

The development of SEMCO [27] (System and software Engineering for embedded systems applications with Multi-COncerns) was started in 2010 by Dr. Brahim Hamid. The SEMCO foundation is a model-based repository of models and patterns and thereby a pattern-based development methodology. In SEMCO, a pattern is a subsystem dedicated to non-functional aspects, to be specified by a non-functional aspects experts, and reused by domain engineers to improve systems/software engineering facing non-functional requirements. It is a good application and promotion of model-driven engineering. The core of SEMCO is a set of DSMLs, search engines and transformations. The DSMLs are devoted to specify patterns, a system of patterns as a set of models to govern their use, and thereby to organize, analyze, evaluate and finally validate the potential for reuse. Engines allow to find/select these artifacts from a repository and then transform the results towards specific domain development environments such as UML.

2.7.1 Definitions

We define the following:

Definition 1 (Domain) *We define a domain as a field or a scope of knowledge or activity that is characterized by the concerns, methods, mechanisms,... employed in*

the development of a system. The actual clustering into domains depends on the given group/community implementing the target methodology.

Definition 2 (Modeling artifact) *We define a modeling artifact as a formalized piece of knowledge for understanding and communicating ideas produced and/or consumed during certain activities of the system engineering processes. The modeling artifact may be classified in accordance with engineering process levels.*

Definition 3 (Modeling artifact system) *A modeling artifact system is a collection of modeling artifacts forming a vocabulary. Such a collection may be skillfully woven together into a cohesive "whole" that reveals the inherent structures and relationships of its constituent parts toward fulfilling a shared objective.*

Adapting the definition of [65], we propose the following:

Definition 4 (S&D Pattern) *A security and dependability pattern describes a particular recurring security and/or dependability problem that arises in specific contexts and presents a well-proven generic scheme for its solution.*

Definition 5 (S&D Pattern System) *We define a security and dependability pattern system as a modeling artifact system where its constituent parts are security and dependability patterns, its referenced property models and their relationships.*

Definition 6 (Pattern System Configuration) *A Pattern system configuration is one of the possible configurations derived from the pattern system. It represents the structure of an application based on S&D patterns which will be deployed on the platform.*

Based on [53], we define the following:

Definition 7 (Property) *A property is a basic attribute shared by all members of an artifact (Pattern, Resource, etc).*

Definition 8 (Resource) *We define a Resource as a modeling artifact which represents a piece of the platform. It defines a set of parameters that will be later used to automate the analysis of applications based on S&D patterns.*

Definition 9 (Platform) *The platform is defined as a set of interconnected hardware resources on which the software elements can be deployed.*

According to Bernstein and Dayal [9],

Definition 10 (Repository) *A repository is a shared knowledge base of information on engineered artifacts. They introduce the fact that a repository has (1) a* Manager *for modeling, retrieving, and managing the objects in a repository, (2) a* Database *to store the data and (3)* Functionalities *to interact with the repository.*

In the context of TERESA, we propose the following:

Definition 11 (Instantiation.) *An instantiation activity takes a pattern and its related artifacts from the repository and adds it to the end-developer environment. This task enables the pattern to be used while modeling.*

The *Instantiation* activity is composed of the following steps:

1. Define needs in terms of properties and/or keywords,
2. Search a pattern in the repository,
3. Select the appropriate pattern from those proposed by the repository,
4. Import the selection into the development environment using model transformation techniques.

Definition 12 (Integration.) *An integration activity happens within the development environment when a pattern and its related artifacts are introduced into an application design. Some development environments may come with native support for the integration.*

2.7.2 SEMCO

The SEMCO approach is a federated modeling framework built on an integrated repository of metamodels to deal with system engineering. The end-user part of such a framework is an integrated repository of modeling artifacts to be used in order (i) to model several concerns of embedded systems engineering: extra functional properties; (ii) to model systems parts: logical software, hardware components and infrastructure. The main goal of SEMCO is to deal with multi-concerns embedded system engineering for several domains.

We build on a theory and novel methods based on a repository of models which (1) promote engineering separation of concerns, (2) supports multi-concerns, (3)

use model libraries to embed solutions of engineering concerns and (4) supports multi-domain specific process. This framework is threefold: providing repository of modeling artifacts, tools to manage these artifacts and guidelines to build methodologies for system engineering.

As shown in Figure 2.5, SEMCO foundation is a federated DSL processes working as a group on how relevant each one is to the key concern. A DSL building process [2] is divided into several kinds of activities: DSL definition, transformation, consistency and relationships rules, design with DSL and Qualification. The three first activities are achieved by the DSL designer and the two last activities are used by the final DSL user.

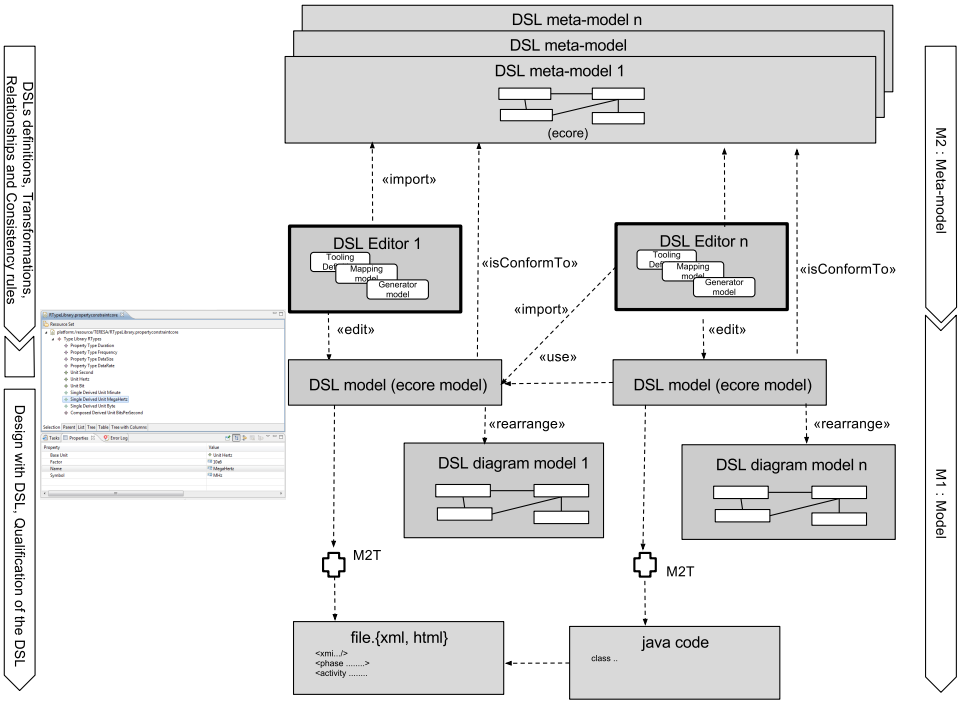

Figure 2.5: SEMCO DSL Building Process and Artifacts

Figure 2.6 illustrates the use of the Eclipse Modeling Framework (EMF) to support the SEMCO process to create our tool suite.

[2] DSL building process defines how development projects based on DSL are achieved.

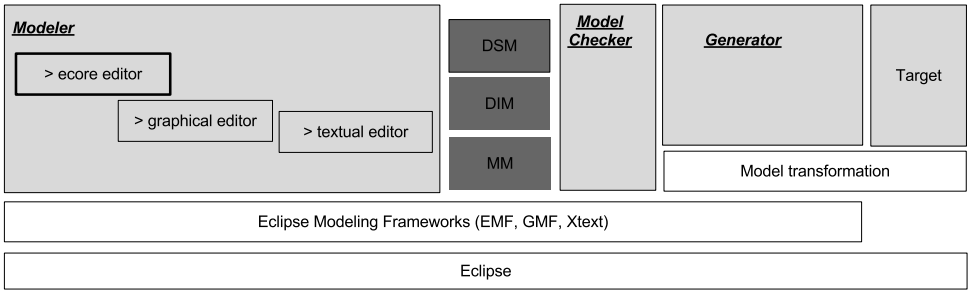

Figure 2.6: Overview of the SEMCO tool suite architecture

2.8 Introduction to the Case Study: Railway Control System (Safe4Rail)

Safe4Rail is in charge of the emergency brake of a railway system. Its mission is to check whether the brake needs to be activated. Most important, the emergency brake must be activated when something goes wrong. The conceptual view of the application is presented in Figure 2.7.

There is a family of products in the railway sector, namely, regional trains, tramways, high speed trains, etc. They share common and variable parts. For instance, consider the calculation of the actual speed and position by Safe4Rail. Their implementation vary among each product type. This mainly depends on the safety level to meet, but also on the type and the number of sensors and actuators involved. These considerations greatly influence how each product is to be implemented, since several issues shall be considered: the number of channel redundancy, the diversity of the channels, the monitoring of the channels, and the interaction with assorted data (type, weight, etc.).

A specialized embedded system was designed to meet stricter safety regulations. In this case, SIL4[3] level is targeted. A number of design techniques from S&D are used, namely redundancies, voting, diagnostics, secure and safe communications. A very strict engineering process was followed, where specific activities were performed in order to achieve certification using the presented approach. For instance, the design of a set of patterns, libraries of S&D and resource properties in order to populate the repository and then to integrate the repository in the aforementioned engineering process.

As depicted in Figure 2.7, the entire system is composed of:

[3]Safety Integrity Level 4

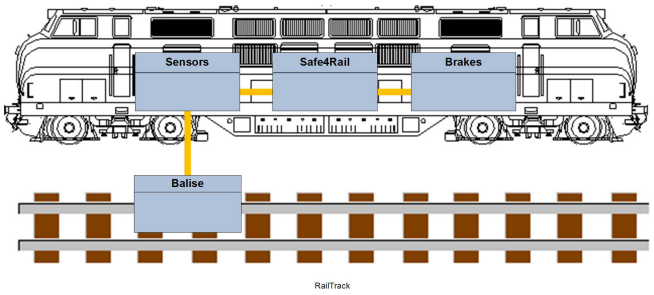

Figure 2.7: Railway Control System

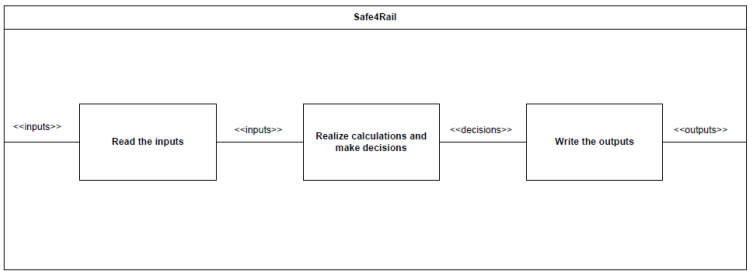

Figure 2.8: Safe4Rail Application

- (1) *brake system.* Stops or decelerates the train. There are several types of brakes: mechanical, electrical and emergency.
- (2) *the system.* Collects inputs from sensors, performs a number of calculations to make decisions, which are typically propagated as outputs.
- (3) *sensor.* Provides inputs in diverse forms, so that decisions can be taken. For instance, there are track tags and radars.
- (4) *railtrack.* The railway convoy moves along the railtrack and the system monitors a number of parameters from assorted sensors.

The fundamental functionality of the system is based on a set of inputs from the railtrack, assorted sensors, balises and so on. Starting from them, it performs some calculations and decides whether the emergency brake needs to be activated. An output signal is sent accordingly.

The Safe4Rail system (see Figure 2.8) realizes the following functionalities:

- (1) reads the inputs (collects the inputs)
- (2) realizes calculations and makes decision (functional code)
- (3) activates break (writes outputs)

Pattern	Origin
TMR	IEC-61508-2, EN-50129
Secure Communication	SSL or its update named TLS proposed in RFC 2246
Majority Voter	IEC-61508-2
Data Agreement	Book "Real-Time Systems: Design Principles for Distributed Embedded Applications"
Safety communication	IEC-61508-2
Watchdog	EN-50126, IEC-61508-3
Reciprocal Monitoring	IEC-61508-2, EN-51028 (Fault Detection & Diagnosis)

Table 2.1: The used patterns in Safe4Rail application

Figure 2.9: Railway demonstrator

The following table shows the list of patterns that are going to be used in the railway demonstrator. Appendix B provides the full description of these patterns.

The hardware platform associated to the Safe4rail application is shown in Figure 2.9. The platform is composed of a carried board on which is installed a conga-CA board with a microprocessor (Intel Atom Z530), a RAM (DDR2 RAM) and a set of interfaces for the connections. The conga-CA is accompanied with an additional programmable resource calculation unit (FPGA-Spartran).

2.9 Conclusion

MDE promotes models as a first class elements. A model can be represented at different levels of abstraction and the MDE vision is based on (1) the metamodeling techniques to describe these models and (2) the mechanisms to specify the relations between them. Model exchange is within the heart of the MDE methodology as well as the transformation/refinement relation between models. MDE frameworks may help software engineering specialists in their tasks, but indeed it would be interesting to provide (partial) solutions and to guide them fulfilling recurring requirements. In software engineering, *Pattern* meets this need. We leverage on this idea to propose a novel common pattern modeling language.

In our work, we promote a new discipline for system engineering using a pattern as its first class citizen: Pattern-Based System Engineering (PBSE). PBSE addresses challenges similar to those studied in software engineering focusing on patterns and from this viewpoint addresses two kind of processes: the process of *pattern development* and *system development with patterns*.

In this thesis, we propose to model and to analyze S&D applications for RCES using pattern as its first class citizen. The integration of S&D features requires the availability of both application domain specific knowledge and S&D expertise at the same time. Currently, the integration of S&D mechanisms is still new in many domains (e.g. smart metering or home control), hence embedded system developers usually have limited S&D expertise. We propose an integrated, MDE based tool-supported approach for capturing and providing this expertise by the mean of S&D patterns.

The proposed approach is based on (i) a model-based repository of S&D patterns, (ii) resource models, and (iii) property models. The modeling framework allows to specify properties, resources, patterns and a model-based repository independently from target development applications and execution platforms. It has been applied to an industrial resource-constrained embedded system (RCES) in the context of the TERESA project.

Chapter 3

Related Work

3.1 Introduction

The requirement for higher quality and seamless development of systems is continuously increasing, even in domains traditionally not deeply involved in such issues. Nowadays, many practitioners express their worries about current software engineering practices. New recommendations should be considered to ground this discipline on two pillars: solid theory and proven principles. We took the second pillar towards software engineering for embedded system applications, focusing on the problem of the specification and the packaging of software modeling artefacts to foster reuse.

In our study, we promote the use of S&D patterns as sub-systems for building the software architecture of the application (Sw) on the one hand, and the use of resource models for building the hardware architecture of the platform (Hw) on the other hand. In order to select the appropriate set of patterns for a given platform, non-functional properties may be used to point out the link between a pattern and its required resources. Repositories of modeling artefacts have gained more attention recently to enforce reuse in software engineering. In fact, repository-centric development processes are more and more adopted in software/system development, such as architecture-centric or pattern centric development processes.

In this chapter, we review exiting work related to the use of modeling techniques for the representation of resources during the specification of hardware platforms, the design of properties on these resources, and how analysis techniques may help to take into account these descriptions during the design of the software part of an embedded system. Moreover, we study pattern modeling in the context of S&D and RCES. Finally, we consider the packaging of the modeling artifacts to enforce reuse

by design.

3.2 Modeling Languages for Resources and Non-Functional Properties

In order to specify platform resources and non-functional properties, the relevant proposals consist of works, from standards and academics work.

3.2.1 Standards

First, we outline some of the most important standards that rely on the modeling and analysis of resources and their properties in embedded systems. Some of the most important works in this direction are those proposed in the context of the three standards : MARTE [53], SysML [54] and AADL [62]. These standards allow the modeling of real time embedded systems by the specification of both the software and the hardware parts and the description of different processes of systems.

MARTE. The recent profile, MARTE (Modeling and Analysis of Real-Time and Embedded systems) [53], which emerged from the UML/SPT profile [51] is a dedicated framework for modeling and analysis concurrency, resources, timing concepts and non-functional properties of embedded real-time systems. The UML MARTE profile provides a set of stereotypes and tag values that can be used for annotating the model elements and for performing analysis. It provides the Generic Resource Modeling (GRM) sub-profile for platform-based modeling and a high level concepts for specifying resource usage. GRM is refined in Software Resource Model (SRM) and Hardware Resource Model (HRM) dedicated to describe software and hardware computing platforms, respectively.

SRM [75] is dedicated to characterize Real-Time multitasking execution platforms by providing modeling artifacts to describe software execution platform covering main multitasking capabilities such as real-time language libraries and real-time operating systems. This is to allow the specialisation of generic resource concepts to software domain, to describe concurrency support (e.g. tasks, interrupts and alarms), to detail interactions between concurrent resources (e.g. messaging, synchronization and mutual exclusion mechanisms) and to depict the software resource brokers (e.g. driver and scheduler).

HRM [69] is a sub-profile for detailed hardware modeling which is composed of two views, a logical view that classifies hardware resources depending on their functional properties, and a physical view that concentrates on their physical nature.

In order the model non-functional properties, MARTE provides the NFP sub-profile for the specification of non-functional properties. NFP provides the capability to describe various kind of values related to physical quantities, such as Time, Mass, Energy. These values are used to describe the non-functional properties of a system.

SysML. SysML (System Modeling Language) [54] is a UML profile "for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities". The so-called Block concept is the common conceptual entity that factorizes many different kinds of system elements such as electronic or software components, mechanical parts, information units and the description of different processes of systems. Blocks articulate a set of modeling perspectives enabling separation of concerns during systems design. However, SysML does not allow the strict modeling of temporal constraints and the resources management.

AADL. Another approach is AADL (Architecture Analysis and Design Language) [62]. It defines an interface for each component and it allows the separation between the implementation of a component and the description of its properties. Note, however, that most AADL properties can find their equivalences in MARTE. AADL describes both the software and the hardware parts of a system and allows the developer to concentrate only on the application level. For example, the developer presents only the threads, processes, and sub-programs of his application. He is not interested on the processing of threads or the communication of components. In addition, AADL addresses platform parameterization and code generation.

The goal of these standards is to propose a common modeling framework to design embedded systems. However, they do not offer same constructs for modeling (because of the varying nature of the disciplines involved in embedded systems design) and especially do not completely satisfied our needs for modeling resources and non-functional properties and helping engineers to model their system without getting bogged down in the details of the proposed standards. To get the best of each standard we made a synthesis of several criteria. The next table summarizes the comparison made on the studied standards. The X axis contains the different criteria taken into account for the comparison, while the first column shows the three

standards : MARTE, SysML and AADL. The sign ✓means that the chosen criterion is satisfied by the selected standard, while the sign ✗means that the criterion is not satisfied.

Standards	Non-Functional Properties	Predefined NFP (Libraries)	NFP independence of resources	System Design	HW/SW Execution Platform	Metamodel	UML profile	Textual DSL
MARTE	✓	✗	✗	✓	✓	✓	✓	✗
SysML	✓	✗	✗	✓	✗	✓	✓	✗
AADL	✓	✓	✓	✓	✓	✓	✗	✓
	Concern			Level of abstraction		Representation		

Table 3.1: *Comparative Table of Embedded Systems Modeling Standards*

Comparison. We selected the comparison criteria under three views: Concern, Level of abstraction and Representation.

1. Our concern in this work is to provide a language for them modeling of platform resources and non-functional properties. The properties may be used to annotate the platform resources in order to specify and analyze them.

2. We are interested in two levels of abstraction, system modeling (i.e. without taking into account the software and hardware specifications) and the detailed modeling of both software (application and software execution platform) and hardware (hardware execution platform).

3. The aim of this work is to provide languages for RCES modeling as reusable and composable models conforming to the meta-model. With the latter, several implementations can be proposed for different technologies, for example we could define a new UML profile which could be used with both MARTE and SysML profiles, or we can define our own DSL based on XML, Eclipse Ecore or other technologies.

3.2.2 Academic Work

Ammar et al. present in [6] a tool extension for resource modeling within UML-based simulative environments. For this, UML notation is extended with new stereotypes that allow the representation of resource related items (such as CPUs, disks, etc.). The software architecture and the resources that the software components require are both represented in the same capsule diagram, which is split in two parts: the software side and the resource side. Capsules are in both sides, but while the ones in the software side represent software components, the resource side capsules represent the resources that the considered architecture may need. This approach aims at migrating the resource representation into the software model notation. This keeps the modeling process unchanged, and also avoids a (probably heavy) transformation procedure to generate the performance model.

In [61], the authors extend UML through a UML profile for modeling non-functional requirements and their dependencies to enable performing trade-off analysis among them. The proposed profile may be used to annotate model elements (UML Classes, UML Components, etc) with necessary information and then calculate satisfaction values of non-functional requirements using model transformation technique.

Based on the study conducted in [79], the authors propose in [78], a language for resource modeling and analysis of embedded systems. The model, called REMES (REsource Model for Embedded Systems), is based on a notion of resources that are characterized by their discrete or continuous nature, the way they are consumed and/or allocated and released, and whether they can be referred to, or not. A number of generic resources can be modeled using this framework, including memory, ports, energy, CPU and buses. The REMES model focuses on component-based behavioral modeling. For that it provides a graphical behavioral language, timed and hybrid automata and Statecharts to express resource usage in a system. REMES provides a rich and powerful framework for formal analysis to address the lack of formal description in some UML-based frameworks.

In [8], Baum et al. present a structured approach to describing resource-usage scenarios. As prerequisite, they classify resource types in two classes: timed-shared and space-shared. Authors argue that a technique for modeling resource-usage scenarios generally has to consider three description aspects: service requirements, service provision and resource interaction. Service provision captures the characteristics of the services offered by the resource, whereas service requirements describe

the resource's demands. Finally, resource interaction links service requirements with provisions. Both a graphical and a textual notation were developed to describe resource-usage scenarios in a structured way. However, such a modeling is limited in terms of resource types and does not take into account other ones like energy resources or communication resources which cannot be clearly characterized.

3.3 Pattern Modeling and S&D Concern

The concept of pattern was first introduced by Alexander [5]. In software engineering, a pattern deals with a specific, recurring problem in the design or implementation of a software system. It captures expertise in the form of reusable architecture design themes and styles, which can be reused even when algorithms, component implementations, or frameworks cannot. With regard to S&D aspects, Yoder and Barcalow [81] were the first to work on security pattern documentation. Many contributions on S&D patterns can be found in literature [18, 24, 76, 81, 82].

Design patterns are a solution model to generic design problems, applicable in specific contexts. Since their appearance, and mainly through the work of Gamma et al [23], they have attracted much interest. Supporting research includes domain patterns, pattern languages and their application in practice.

Pattern Modeling Languages

To give a flavor of the improvement achievable by using specific languages, we look at the pattern formalization problem. *UMLAUT* was proposed by [3] as an approach that aims to formally model design patterns by proposing extensions to the UML metamodel 1.3. They used the OCL language to describe constraints (structural and behavioral). These constraints are defined on metamodels of specified UML elements in the form of meta-collaboration diagrams. Mechanisms of association of these meta level diagrams to their instance level (instances of design patterns) are then defined. This allows to model design patterns accurately in UML. This work is illustrated through two examples of design patterns: visitor and observer.

In the same way, Kim et al. [16] presented *RBML (Role-Based Metamodeling Language)*. RBML is able to capture various design perspectives of patterns such as static structure, interactions, and state-based behavior. This language is based on metamodeling of design patterns and offer three specifications: Structural, Behavioral and Interactive. Each one is characterized by a kind of RBML metamodel: (1)

SPS (Static Pattern Specifications) is a specification of structural design patterns which allows to express the static view, (2) IPS (Interaction Pattern Specification) represents the design pattern in terms of possible interactions between different roles, (3) SIMP (StateMachine Pattern Specifications) can add a behavioral view point to describe the various states in which it may lie in its execution.

Another issue raised in [19] and [17] is visualization. Eden et al. [19] presented a formal and visual language for specifying design patterns called *LePUS*. It defines a pattern in an accurate and complete form of formula with a graphical representation. A diagram in LePUS is a graph whose nodes correspond to variables and whose arcs are labeled with binary relations. With regard to the integration of patterns in software systems, the *DPML (Design Pattern Modeling Language)* [17] allows the incorporation of patterns in UML class models.

S&D Patterns

Several approaches exist in literature targeting S&D design patterns [18, 24, 76, 81, 82]. They allow to solve very general problems that appear frequently as sub-tasks in the design of systems with security and dependability requirements. These elementary tasks include secure communication, fault tolerance, etc. The recently completed *FP6 SERENITY* project has introduced a new notion of S&D patterns. SERENITY's S&D patterns are precise specifications of validated S&D mechanisms, including a precise behavioral description, references to the provided S&D properties, constraints on the context required for deployment, information describing how to adapt and monitor the mechanism and trust mechanisms. The S&D SERENITY pattern is specified following several levels of abstraction to bridge the gap between abstract solution and implementation – S&D classes, S&D patterns and S&D implementation. Such validated S&D patterns, along with the formal characterization of their behavior and semantics, can also be basic building blocks for S&D engineering for embedded systems. [66] explains how this can be achieved by using a library of precisely described and formally verified security and dependability solutions, i.e., S&D classes, S&D patterns, and S&D integration schemes. The work of [24] reports an empirical experience, about the adopting and eliciting S&D patterns in the Air Traffic Management (ATM) domain, and show the power of using patterns as a guidance to structure the analysis of operational aspects when they are used at design stage. A survey of approaches to security patterns is proposed in [82].

To summarize, in software engineering, design patterns are considered effective tools for the reuse of specific knowledge. However, a gap between the development of systems using patterns and the pattern information still exists. This becomes even more visible when dealing with specific concerns namely security and dependability for several application sectors in RCES. From the pattern-based software engineering methodological point of view, few works are devoted to this concern. They are in line for the promotion of the use of patterns in each system/software development stage. However, existing approaches using patterns often target one stage of development (architecture, design or implementation) due to the lack of formalisms ensuring both (1) the specification of these artifacts at different levels of abstraction, (2) the specification of relationships that govern their interactions and complementarity and (3) the specification of the relationship between patterns and other artifacts manipulated during the development lifecycle and those related to the assessment of critical systems.

3.4 Repository

This section aims to present related work on the repository concept in order to determine the most appropriate structure for the TERESA repository. The repository concept is used in different research fields. Among them we can distinguish: model repository, pattern repository, software repository, repository of software components, ontology repository for the semantic web, web services repository, etc. In our context, we focus on the review of the most relevant works: model repository and pattern repository.

3.4.1 Repository of models

In Model-Driven Development (MDD), model repositories [9, 21, 44] are used to facilitate the exchange of models through tools by managing modeling artifacts. For instance, as presented in the ebXML specifications [48] and the ebXML Repository Reference Implementation[1], a service repository can be seen as a metadata repository that contains metadata on location information to find a service. In [44], the authors propose a reusable architecture decision model for setting-up model and metadata repositories. In addition, some helpers are included in the product for selecting a

[1] http://ebxmlrr.sourceforge.net/

basic repository technology, choosing appropriate repository metadata and selecting suitable modeling levels for the model information stored in the repository.

The ReMoDD (Repository for Model Driven Development) project [21] focuses on MDD for reducing the effort of developing complex software by raising the level of abstraction at which software systems are developed. This approach is based on a repository containing artifacts (e.g. documented MDD case studies, modeling exercises and problems) that support research and education in MDD. Another issue is graphical modeling tool generation as studied in the GraMMi project [63]. GraMMi's Kernel allows to manage persistent objects. The kernel aims at converting the objects (models) in an understandable form for the user via the graphical interface. Recently, the MORSE project [36] proposes a Model-Aware Service Environment repository addressing two common problems in MDD systems: traceability and collaboration.

3.4.2 Pattern Repository

Patterns are stored in repositories to comprehensibly explain their classification. The organization of patterns in a repository allows to discover the relationships among them and to facilitate the selection of the most appropriate ones. The repository should have a structure in order to optimize the accesses (selecting patterns with criterion's and publishing new patterns into it). Finding the appropriate pattern to solve a particular security or/and dependability problem is difficult because of the lack of a scientific classification scheme for S&D patterns. In the following we discuss some work related to such a problem: classification schemes to help in finding the appropriate pattern.

Some classifications are based on security concepts. For example, ISO/IEC 13335 [39] provides a definition of the five key concepts: security, confidentiality, integrity, availability and accountability. A pattern classification scheme based on these domain level concepts, will facilitate pattern mining and pattern navigation. An implicit culture approach for supporting developers in choosing patterns suitable for a given problem is described in [11]. In this vision, the repository contains patterns that are selected depending on the history of their use regarding decisions made by other developers to deal with related problems. In [60], a mathematical structure is proposed for organizing patterns depending on several categories. An ontological approach for selecting design patterns is proposed in [25] to facilitate the understanding and reuse during software development. In their paper, the authors present an

ontology which describes the design pattern format and their relationships. They use a pattern system/language in order to facilitate the design, integration, selection and reuse of design patterns.

Most existing classifications in the literature [41] are based on:

- Applicability is used to protect resources against unauthorized use, disclosure or modification. In addition, applicability is used to make predictable and uninterrupted access to resources or services.
- Product and process (structural and procedural)
- Logical tiers (a) web: this tier takes into account the external requests, authentication and authorization (b) business: this tier takes into account the security services in the business like RBAC (c) integration: this tier facilitates secure integration with external data sources
- Security concept (confidentiality, integrity, availability and accountability) – see ISO/IEC 13335-1 and ISO 7498-2 [37, 38].

Another aspect that has been considered is system viewpoints. Based on the idea of the Zachman Framework [83] (classification based on system viewpoints and interrogatives) the Microsoft Patterns and Practices group Classification [77] distinguishes the following elements: (a) Merits (clearly identifies the context of each pattern, help to identify missing patterns), (b) Flaws (more dedicated to functional patterns – non-functional patterns tend to cover many levels of system development and also many interrogatives), (c) Improvement (add icons in each pattern to provides classifications).

[80] presents a survey of business process model repositories and their related frameworks. This work deals with the management of a large collections of business processes using repository structures and providing common repository functions such as storage, search and version management. It targets the process model designer allowing the reuse of process model artefacts. A comparison of process model repositories is presented to highlight the degree of reusability of artifacts. The metamodel described in our work may be used to specify the management and the use of this kind of process models. In fact, a process model aspect or the process model as a whole can be seen as artifacts supported by our metamodel. In return, the vision of the business process model repositories may be used in our work to manage the process element type libraries.

3.5 Conclusion

In this chapter we studied the relevant works and practices related to our contributions. We subdivided this study into three sections dealing with the resources and non-functional properties modeling, S&D patterns modeling and model-based repository modeling, receptively.

We identified certain gaps in the previous works, especially, those dealing with the adaptation of S&D patterns for RCES application modeling, taking into account the strong link between the S&D patterns-based applications and the platform resources consumption. In addition, we identified lacks on the reusing techniques of the modeling blocs, although, the previous works concentrate mainly on the modeling phase. Furthermore, the repository of models proposed previously are almost dedicated while there is no approach for model-based repository, in other words, there is no methodology for repository generation for different domains and uses.

In our work, different levels of abstraction are used to get a common representation of patterns for several domains. In the TERESA concern, systems include a combination of hardware and software components. This may add some difficulties to build a simple modeling framework. A high level of abstraction is proposed to represent S&D patterns to capture several aspects of security and dependability in the different domains of RCES, not an implementation of a specific solution. Other issues are:

- In order to integrate a pattern in a system (application), some significant additional information about the pattern is required. For instance, the resource consumption, interfaces and their requirements. The goal is to capture how the system interacts with the patterns, and how the structure of the pattern interacts with other patterns in the case of system of patterns.

- What kind of information should be used? Especially when dealing with software and hardware components ?

The main goal of a repository is to store data and to offer a set of actions to interact with it. Most of the time a repository has to provide the following actions: store files, authentication and access control, check-out/check-in files, file versioning, file metadata storage and data search. Some of the classical formats used are XML, XMI and XSD. Then the graphical view of artifacts can be made by using some format transformations. This one can be realized thanks to XSLT (eXtensible Stylesheet Language Transformation) transformations to ease the reuse of XML

solutions. On another hand, XBRL (eXtensible Business Reporting Language), an XML format, also allows the interoperability of information. Thus, an XBRL file can be converted to a standard format such as HTML ASCII and also PDF, which will provide an ergonomic aspect.

In the TERESA approach, the following questions arise about the specification of the repository:

- What kind of visualization interfaces to use?
- What kind of interaction interfaces to use? (some information about how artifacts are published and how artifacts are queried must be provided by the repository)
- How to organize the data ?
  - what kind of metadata to use to ease the selection of data?
  - what kind of data structure to use?
  - what kind of use is supported? (static and/or dynamic). How to store the data ?
  - what kind of format to use in order to store the data?

Chapter 4

Contribution to the Modeling of S&D Applications for RCES

4.1 Introduction

The proposed approach promotes model-based development coupled with a repository of modeling artifacts, including S&D property models, resource models and S&D patterns. This approach aims to define an engineering discipline for S&D application that is adapted to resource-constrained embedded systems. The main goal of this chapter is to define a modeling framework to support the specifications and the definitions of these modeling artifacts (***RG1***).

We begin with the presentation of the development framework to elucidate the purpose and the use of the metamodel. Hence, we clarify the development context where this metamodels fit by articulating a methodological view for a development process emphasizing models and transformations. Then, we present in detail the concepts as the basis for the definition of the modeling languages. Finally, we highlight the associated pattern system configuration management, analysis framework and how they will be used in an MDE approach. Moreover, we provide a set of transformations that can be useful in the analysis of software architecture based on patterns. The next chapter details the specification models for the model-based repository and its APIs.

4.2 Repository-centric Resource-aware System and Software Engineering

Security and Dependability are not building blocks added to an application at the end of the life cycle. It is necessary to take into account these concerns from the requirement to the integration phases. We promote a new discipline for system engineering using a pattern as its first class citizen, towards meeting our wider objective: Pattern-Based System Engineering (PBSE). PBSE addresses challenges similar to those studied in software engineering. Closely related to our vision is the Component Based Software Engineering (CBSE)[14, 15, 68]. Therefore, PBSE focuses on patterns and from this viewpoint addresses two kind of processes: the process of pattern development and the process of system development with patterns. The main concern of the first process is designing patterns for reuse and the second one is finding the adequate patterns and evaluating them with regard the system-under-development's requirements.

In the process model visualized in Figure 4.1, the developer starts by the system specification fulfilling the requirements. In a traditional approach (non pattern-based approach) the developer would continue with the architecture design, module design, implementation and test. In our vision, instead of following this phases and defining new modeling artifacts, that usually are time and efforts consuming as well as errors prone, the system developer merely needs to select appropriate patterns from the repository and integrate them in the system under development.

Once the repository[1] is available, it serves an underlying engineering process for S&D application development. Figure 4.1 sketches the roles of the repository, models and transformations. Each model layer corresponds to a distinct level of abstraction. The usage of this framework proceeds as follow. The developer creates a model representing the target platform importing appropriate resource models from the repository. For the software part, the system developer executes the search/select actions from the repository to import patterns building a pattern system configuration. For each of these configurations, a mapping through an allocation process is executed. The allocation supports the links between a pattern and its required resources from the platform model. The result of the allocation, as the platform model and a configuration for a model of a system of patterns, is then used for analyzing the resource consumption of the patterns with regard to the resources defined

[1] The repository system populated with S&D patterns and models

in the platform model (e.g. memory, processing power). The result of the analysis is then delivered for evaluation. The developer can then use MDE techniques, such as refinement, for implementing the system using the appropriate pattern system configuration.

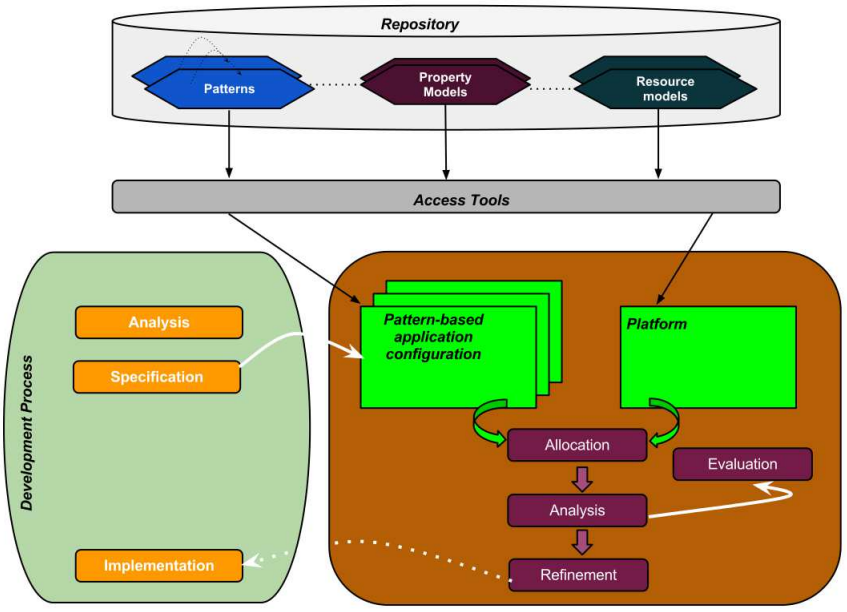

Figure 4.1: Resource-aware System and Software Engineering

4.3 Artifacts Modeling Languages

To foster reuse of patterns in the development of critical systems with S&D requirements, we are building on a metamodel for representing S&D patterns in the form of subsystems providing appropriate interfaces and targeting S&D properties to enforce the S&D system requirements. Interfaces will be used to exhibit the pattern's functionality in order to manage its application. In addition, interfaces supports interactions with security primitives and protocols, such as encryption, and specialization for specific underlying software and/or hardware platforms, mainly during the deployment activity. As we shall see, S&D and resource models are used as model libraries to define the S&D and resource properties of the pattern.

4.3.1 A Metamodel for Non-Functional Properties (GPRM)

GPRM is a metamodel defining a new formalism (i.e. language) for describing property libraries including units, types and property categories. The following

paragraph details the meanings of the principal classes of the *GPRM Metamodel*, which is depicted with Ecore notations in Figure 4.2.

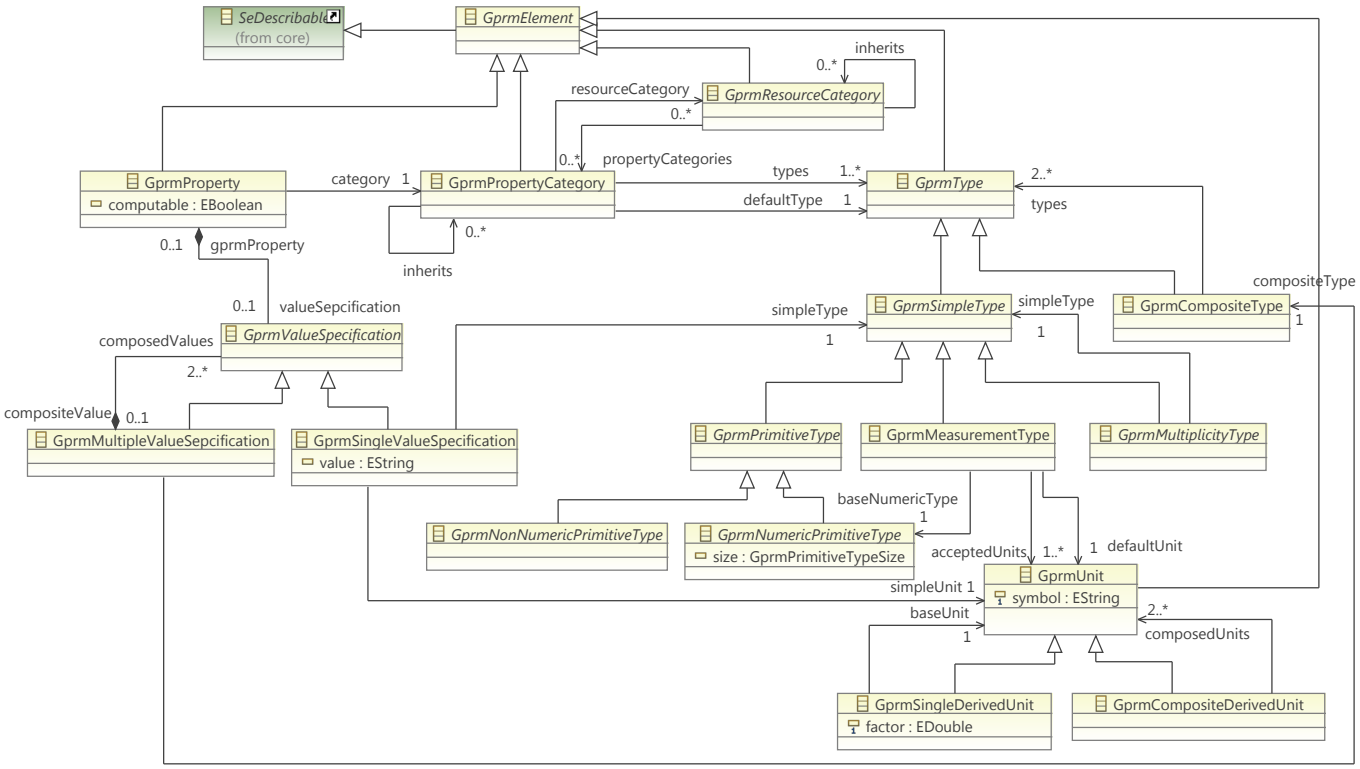

Figure 4.2: The (simplified) GPRM Metamodel

- *GprmElement* is an abstract concept that represents an element of GPRM model. It combines the attributes needed to identify (ID) and to describe (name and description) all elements of a GPRM model.

- *GprmProperty* A property is a basic attribute shared by all members of an artifact (Component, Pattern, Resource, etc). The property is defined by its category, its type and its value. According to the describable artifact, the value of the property may be set by the designer during the design activity or calculated during the analysis activity. In the first case, the property is used to describe a characteristic of the artifact in a certain design, which does not change depending on the environment. In the second case, the property is mainly used to describe a computable characteristic depending on the environment.
  Example: *CPU execution time for encryption* and*energy consumption for encryption* are two properties (resource-related) of the pattern *SecureCommSSL*.

- *GprmPropertyCategory* A property category is a classification for properties. Its role is to regroup all the properties sharing common characteristics. These characteristics may depend on the user or application domain viewpoint. A category supports a set of types that define the nature of the property, it can also be defined based on other categories by specialization. A category is defined with at least one default type.
  Example: *CPUTime* is a category that includes properties such as *CPU execution time for encryption* and *CPU execution time for authentication*. *PowerConsumption* regroups *energy consumption for encryption* and *energy for authentication* in the same category.

- *GprmType* A type is an abstract concept that supplies the definition of a typed element such as a pattern property. It defines the range of values represented by a typed element. Types may be used to type properties, operation parameters, or other elements of the model.

- *GprmValueSpecification* A value specification is an abstract element that specifies the instances of a typed element according to its definition.

- *GprmSimpleType* is a specialization of GprmType to define an atomic type.

- *GprmCompositeType* is a specialization of GprmType to define a composite type.

- *GprmMeasurementType* is a specialization of GprmSimpleType to define the nature of a physical measurement.
  Example: *Duration* is a type that is used to measure time, and which is used to type properties owned by the category *CPUTime*, *DataSize* meanwhile is used to measure the amount of memory (e.g. *Ramsize*'s type is *DataSize*)

- *GprmUnit* is basic element which is used to measure the magnitude of the quantity (e.g. *timing* properties are measured by *sec, min* and *hr* units).

- *GprmSingleDerivedUnit* is a specification of *GprmUnit* which is used to measure the same nature of quantity than its basic unit (e.g. *Min* is derived from *Sec* which is used for time measurement).

- *GprmCompositeDerivedUnit* is a specification of *GprmUnit* which is used to measure another nature of quantity than its composite units (e.g. *kilometer* is

used to measure length, *hour* measures time, while *kilometer per hour* is for velocity measurement).

Properties Modeling. The main process for building property library is visualized in Figure 4.3, pointing three principal activities:

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

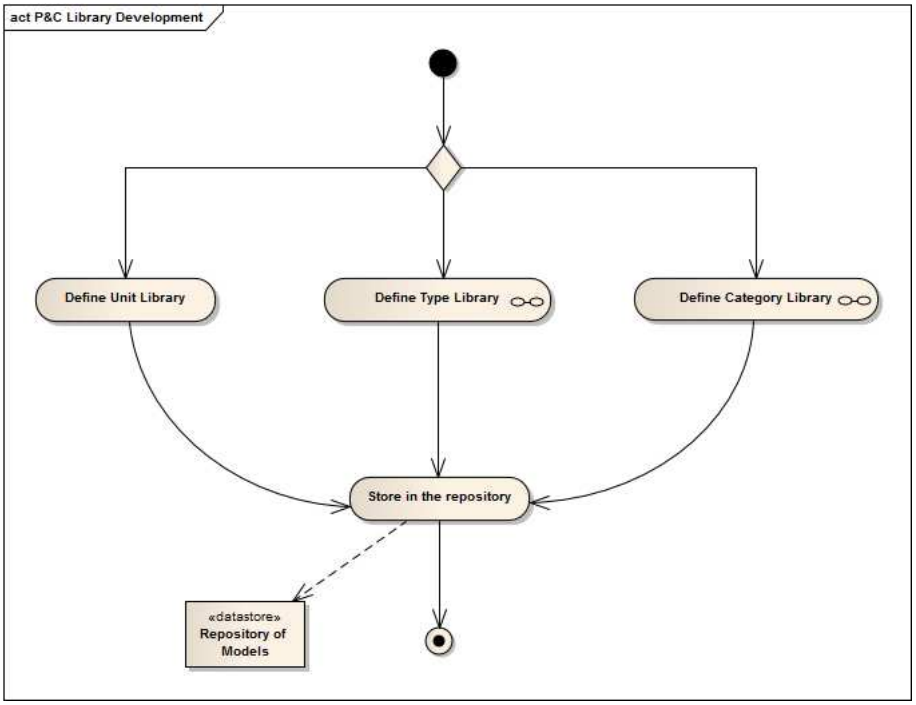

Figure 4.3: Property Library Development

Once the appropriate activity is selected, the left and right part of Figure 4.4 show the process for building a Type Library and a Category Library, respectively.

4.3.2 A Metamodel for Resource (SERM)

In the development context defined in our approach, embedded platforms are seen as a composition of (hardware) resources. Resources are pieces of the platform and can be combined and linked together without having an external observable

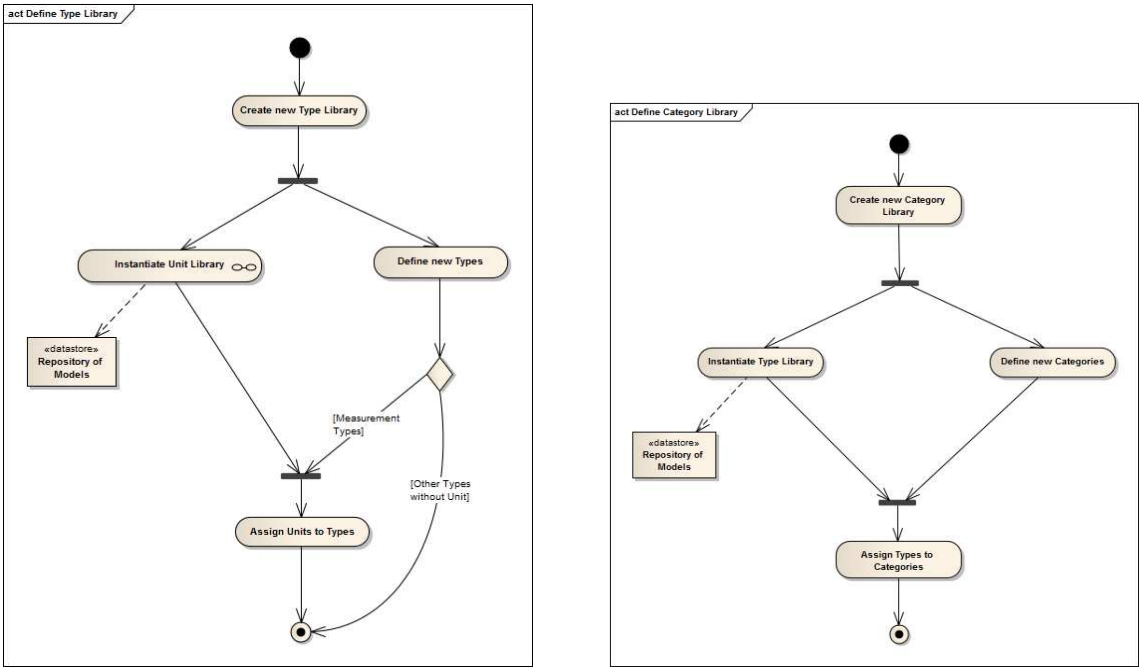

Figure 4.4: Type and Category Libraries Definition Processes

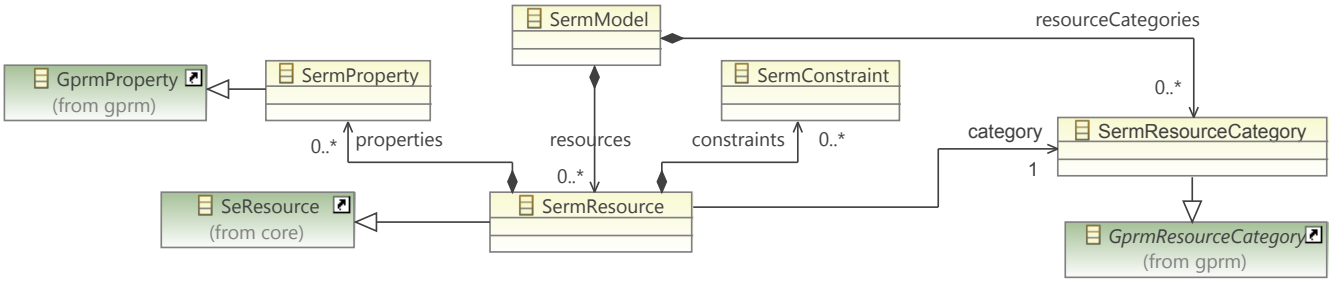

Figure 4.5: The (simplified) SERM Metamodel

state. The *SERM Metamodel* is described with Ecore notations in Figure 4.5 and the meanings of the principal classes are more detailed in the following paragraph.

- *SermResource* is a modeling artifact which represents a piece of the platform. It define a set of parameters that will be later used by the framework to automate the analysis of applications based on S&D patterns.

  Example. *IntelAtom Z530* and *DDR2 RAM* are two hardware resources for computing and data storage, respectively.

- *SermResourceCategory* A resource category is a classification for resources. Its role is to regroup all the resources sharing common characteristics. These

characteristics may depend on the user or application domain viewpoint. The main categories are those related to computing, data storage and to energy consumption. However other categories may be defined such as peripheral, sensor, actuator, etc. A category may also be built based on existing categories by specialization.

Example. As computing resource categories we can define *CPU*, *FPGA*, *DSP*, etc. Therefore, the *IntelAtom Z530* belongs on *CPU* resource category and the *DDR2 RAM* belongs on *RAM* resource category.

- *SermProperty* is a particular characteristic of a resource. So, each property of a resource will be validated at the time of the resource validating process and the result will be compiled as a set of constraints which have to be satisfied by the platform.

4.3.3 A Metamodel for S&D Patterns (SEPM)

The goal of the current section is to propose model-based S&D patterns to get a common representation of patterns for several domains in the context of trusted embedded system applications. The solution envisaged here is based on metamodeling techniques to encode S&D patterns at an even greater level of abstraction. Therefore, a pattern can be stored in a repository and can be loaded according to the desired S&D properties. As a result, S&D patterns will be used as bricks to build trusted applications through a model driven engineering approach.

SEPM is a metamodel defining a new formalism (i.e. language) for describing S&D patterns, and constitutes the base of our pattern modeling language. SEPM describes all the artifacts (and their relations) needed to represent S&D patterns in the context of trusted embedded system applications. Here we consider patterns as building blocks that expose services (via interfaces) and manage S&D and Resource properties (via features) yielding a unified way to capture meta-information related to a pattern and its context of use. Such a pattern is specified by means of a domain-independent generic and a domain-specific representation.

The following paragraph details the principal concepts of the SEPM metamodel to specify an S&D pattern, as described with Ecore notations in Figure 4.6.

- *SepmPattern* represents a modular part of a system that encapsulates a solution of a recurrent problem. An *SepmPattern* is modeled throughout the development life cycle and successively refined into deployment and run-time.

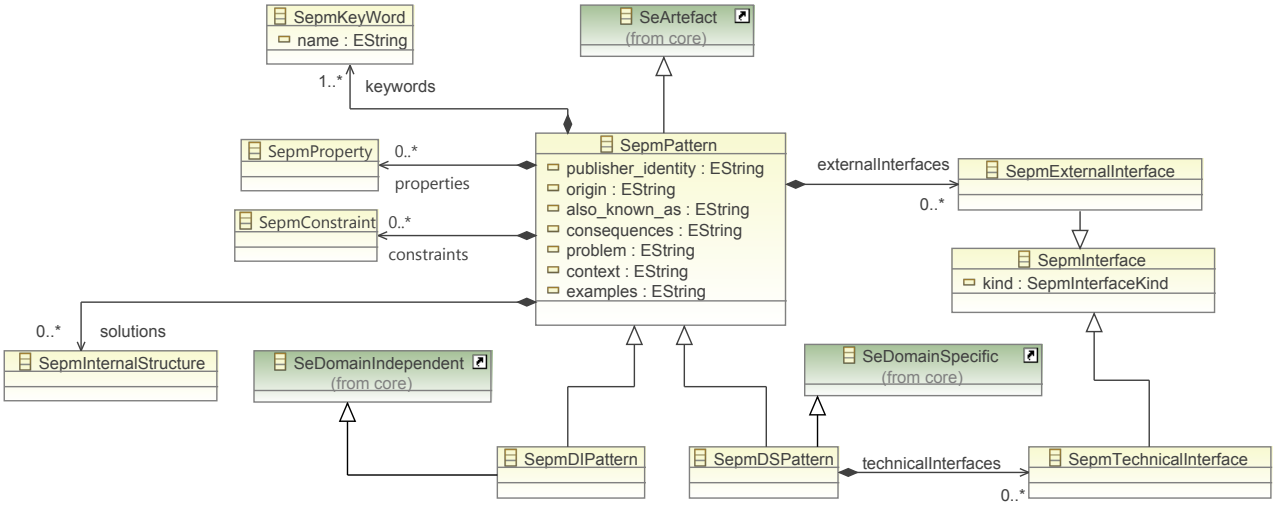

Figure 4.6: The (simplified) SEPM Metamodel

An *SepmPattern* may be manifested by one or more artifacts, and in turn, these artifacts may be deployed to their execution environment. A deployment specification may define values that parametrize the pattern's execution. The *SepmPattern* has some fields that define its identity. These fields are based on the GoF [23] information and are defined as follows:

- *id.* is unique identifier of the pattern
- *name.* is the name of the pattern,
- *problem.* describes the problem solved by the pattern,
- *also known as.* gives a list of names under which the pattern is also known,
- *version.* the version of the pattern,
- *date.* date of creation,
- *publisher identity.* is the identity of the publisher(s),
- *origin.* corresponds to the origin of the pattern – Industrial, Academic ...,
- *consequences.* corresponds to the set of consequences which occur once the pattern is integrated,

- *SepmInterface SepmPattern* interacts with its environment with *Interfaces* which are composed of *Operations*. Larger pieces of a system's functionality may be assembled by reusing patterns as parts in an encompassing pattern

or a composition of patterns, and wiring together their required and provided interfaces. More precisely, an *SepmPattern* owns provided and has required interfaces. A provided interface is implemented by the *SepmPattern* and highlights the services exposed to the environment. A required interface corresponds to services needed by the pattern to work properly. Finally, two kinds of interfaces are considered:

  - *SepmExternalInterface* allows implementing interaction with regard to:
    * *integrating* an *SepmPattern* into an application model. These interfaces are realized by the *SepmPattern*.
    * *composing SepmPattern* together.
  - *SepmTechnicalInterface* allows implementing interaction with the platform. For instance, at a low level, it is possible to define links with a software or hardware module for the cryptographic key management.

- *SepmProperty* is a particular characteristic of a pattern. A *Property* is either an *SDProperty* or a *RProperty*. So, each property of a pattern will be validated at the time of the pattern validating process and the result will be compiled as a set of constraints which have to be satisfied by the platform. This artifact will simplify and enhance the selection/search activities during the pattern-based engineering process.

- *SepmInternalStructure* constitutes the implementation of the solution proposed by the pattern. Thus the *SepmInternalStructure* can be considered as a white box which shows the details of the *SepmPattern*. In order to capture all the key elements of the solution, the *SepmInternalStructure* is composed of two kinds of ***Structure***: ***static*** and ***dynamic***. Please, note that a same pattern could have several possible implementations.

- *SepmDSPattern* inherits from *SepmPattern*. It is used to build a pattern at DSPM. A *SepmDSPattern* has *SepmTechnicalInterface* in order to interact with the platform.

Example: We illustrate the usage of the SEPM for specifying a pattern at domain independent and at domain specific level with the example of secure communication pattern. For the sake of simplicity, we only specify the interfaces and the properties.

For the DIPM, the internal structure representing the pattern solution may be modeled as static diagrams using UML structure diagrams. A subset of the functions provided by the external interfaces are:

- $send(P, ch, m)$: P sends message m to Q on the channel ch,
- $receive(P, ch, m)$: P receives and accepts message m from Q on the channel ch,

with $P, Q \in \{C, S\}$, $ch(C, S) = ch(S, C)$ denoting the communication channel of client (C) and server (S), and m a message. The properties definitions require the availability of the required property libraries. Here, we specify an S&D property: "authenticity of sender and receiver". To type the category of this property we use a category from the earlier defined in the S&D category library: "Authenticity".

The DSPM modeling level is a refinement of the DIPM that considers the specific characteristics and dependencies of the application domain. Different DSPM can refine the same DIPM for different domains. In our example, we use HMAC[2] protocol as a mechanism related to the application domain to refine the secure communication pattern. The internal structure, external interfaces and properties refine the ones defined at DIPM introducing domain specific concepts related to the HMAC mechanism. For example, a function $send(P, ch, m, mac)$ refines the DIPM function $send(P, ch, m)$ adding the mac as the generated message authentication code. A subset of the functions provided by the external interfaces are:

- $send(P, ch, m)$: P sends m to Q on the channel ch,
- $send(P, ch, m, mac)$: P sends m and the corresponding MAC to Q on the channel ch,
- $receive(P, ch, m)$: P receives m on the channel ch,
- $receive(P, ch, m, mac)$: P receives m and corresponding MAC on the channel ch,

with $P, Q \in \{C, S\}$, $ch(C, S) = ch(S, C)$ denoting the communication channel of client (C) and server (S), m a message and mac the generated message authentication code.

In addition to the refinement of the concepts used at DIPM, the DSPM involves developing technical internal interfaces as a set of functions related to the use of

[2]Keyed-Hash Message Authentication Code

HMAC to refine the secure communication pattern. A subset of the functions provided by the technical interfaces are:

- $generateAH(P, keymac, m, mac)$ to prepare an appropriate authentication header (MAC) for the message m.
- $checkAH(P, keymac, m, mac)$ verifies that the message authentication code for m is correct and originates from Q.

with $P, Q \in \{C, S\}$, $ch(C, S) = ch(S, C)$ denoting the communication channel of client (C) and server (S), m a message and mac the generated message authentication code.

The interfaces (external and technical) with the required libraries of properties are then used for developing the pattern properties. Here, in addition to the refinement of the S&D properties identified in the DIPM, at this level we identify some related resource properties, e.g. the size of the cryptographic key.

Pattern Modeling Process. The process'root, as shown in Figure 4.7, indicates the start of the creation of a DIPM pattern interacting with *Pattern Repository*, a data base of informal definitions of patterns. It contains some initialization actions to define the pattern's 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 tool for UML modeling).
- *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. The supporting activities require the interaction with the 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 to the repository.

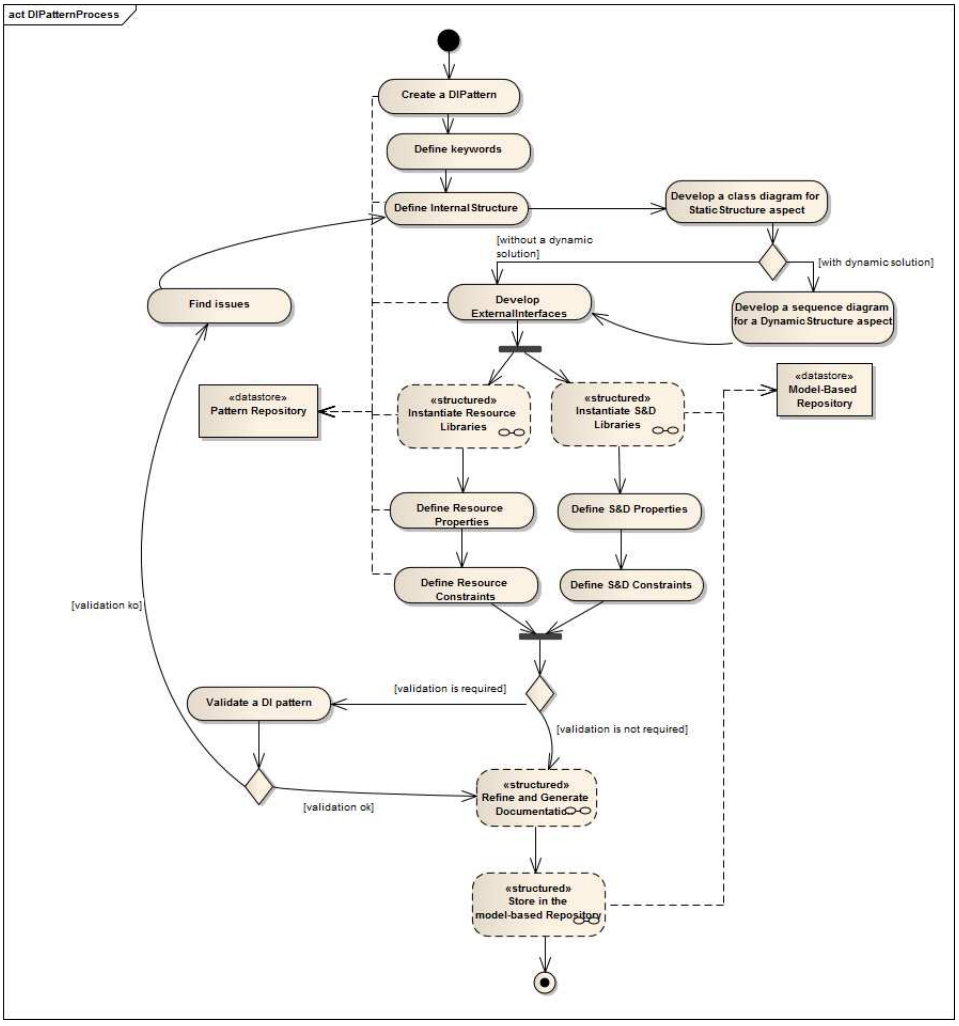

Figure 4.7: Pattern development process at DIPM

A Metamodel of system of patterns. S&D Pattern system is a modeling artifact system where its constituent parts are S&D patterns, its referenced property

models and their relationships. in the following we detail the meanings of the principal classes of the *Pattern system metamodel*, which is depicted with Ecore notations in Figure 4.8.

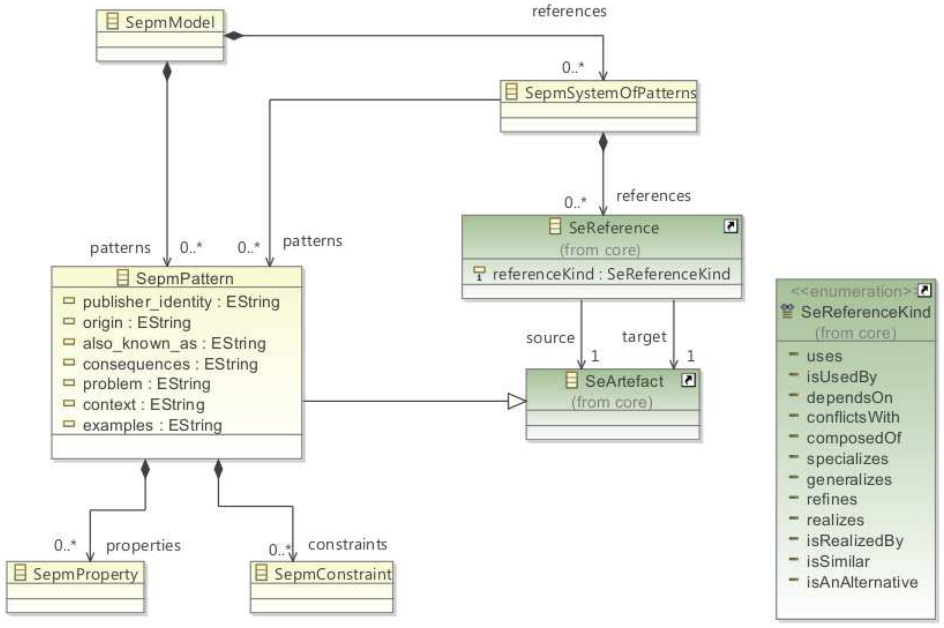

Figure 4.8: A metamodel of patter system

- *SepmSystemOfPatterns.* A system of S&D patterns represents a system of modeling artifacts where artifacts are S&D patterns and links between these artifacts (*SeReference*).
- *SeReferenceKind.* We use an enumeration to define a set of types of links between S&D patterns. In the context of this thesis, we will use the following relationships:
  - *refines.* This link is used to represent the refinement relationship between two patterns.
  - *specializes.* This link is used to represent the specialization relationship (detail).
  - *uses.* This link is used to represent the functional dependency relationship between two patterns.
  - *isSimilar.* This link allows to link two patterns that perform the same functionality. This link is often used to link software patterns to their equivalent hardware patterns.

4.4 Pattern System Configurations Management

In this section we show how we integrate a pattern from the input pattern system in pattern system configuration. We do this using relationships between patterns. A system of pattern is composed of a set of patterns and their relationships (see Definition 5 in Section 2.7.1). Let S be a pattern system. We denote by $P(S)$ and $R(S)$ the set of patterns and the set of the relationships of the pattern system S, respectively As shown in Figure 4.9, we have the "complete system of patterns" in the top of the Figure and we have the "basic configuration of the system of pattern" on the lower part. Let Sc and Sb denote the complete system of patterns and the basic configuration of the system of pattern, respectively. Then, $P(Sc) = \{P1, P1', P2, P2', P3, P3', P4\}$; $R(Sc) = \{IsSimilar, IsAnAlternative\}$ and $P(Sb) = \{P1, P2, P3, P4\}$; where $P1$ and $P2$ are similar to $P1'$ and $P2'$ respectively and $P3$ is an alternative to $P3'$. To integrate a pattern in under-building system of pattern configuration, we propose a simple algorithm called $configurationGen$.

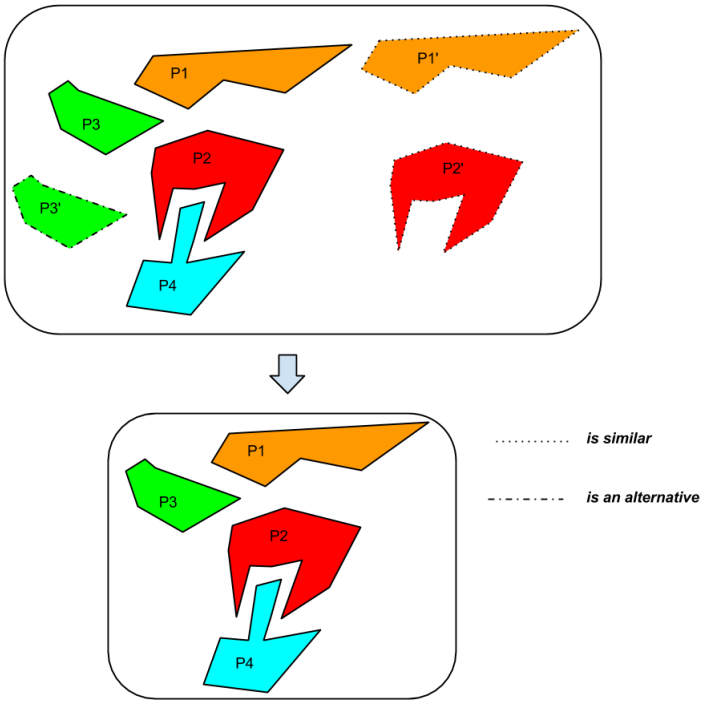

Figure 4.9: Generation of pattern system configuration

This algorithm takes as input Sc and Sb and a referenceKind from $R(Sc)$ to find further configurations which are equal to Sb with respect to the relationship (i.e. referenceKind). The base of the pattern system (Sb) is parsed and for each

pattern in this system equal patterns (with respect to the referenceKind) are looked up in the complete system of patterns. If a pattern is found, a new configuration is generated from the base system and the pattern from Sb is replaced by the equal pattern from Sc. If multiple patterns are equal to a patterns in the base system, multiple configurations (one for each equal pattern) are generated. This algorithm allows to generate equal (e.g. similar, alternative) pattern system configurations and, later on, to analyze these different configurations.

```
1  Algorithm configurationGen
2    Input: Sc model, Sb model, referenceKind.
3    Output: A set of system configurations
4
5    for each Pattern in the Sb model, do
6      for each SeReference in Pattern.references, do
7        if SeReference.ReferenceKind = referenceKind, then
8          duplicateModel(newConfig model, Sb model)
9          replace Pattern by SeReference.referencedPattern from Sc model in newConfig
               model
10         save(newConfig model)
11       endif
12     endfor
13   endfor
```

IsSimilar Configuration

Applying the previous algorithm as $configurationGen(Sc, Sb, IsSimilar)$ we get two possible configurations of systems of pattern S_1 and S_2, where $P(S_1) = \{P1', P2, P3, P4\}$ and $P(S_2) = \{P1, P2', P3, P4\}$, such as visualized in Figure 4.10.

IsAnAlternative Configuration

Applying the previous algorithm as $configurationGen(Sc, Sb, IsAlternative)$ we get the system of pattern configuration S_3 where $P(S_3) = \{P1, P2, P3', P4\}$, such as visualized in Figure 4.11.

4.5 Transformations for Analysis

The analysis in this work implies analyzing the resource consumption and then the impacts of using a certain pattern system configuration in a certain platform. This will help during the pattern selection activity regarding the target platform and assists system designers with awareness about the resource consumption satisfaction

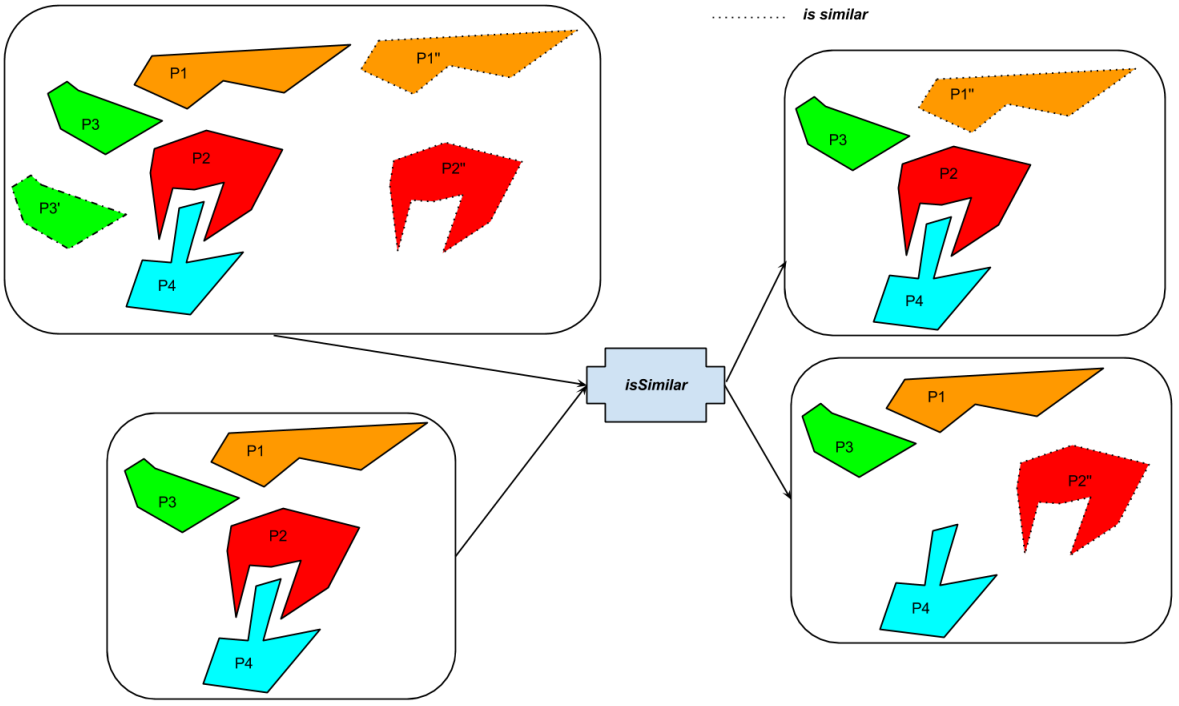

Figure 4.10: Generation of pattern system configuration- IsSimilar

of the patterns. For that, we propose to use model transformations techniques (such as refined model generation and documentation generation) to calculate resource consumption values and to incorporate these information into system models.

The model-to-model transformation for analyzing the platform takes as input the platform model and a configuration for a model of a system of patterns. This transformation parses the used patterns for their consumption of different resources defined in the platform model (e.g. memory, processing power) and adds these up. The result is then injected into the platform model as the computable value of the resource.

The resulted model is then transformed by a model-to-text transformation to visualize the consumption of the pattern system configuration of the different resources of this platform. This transformation takes as input the platform model which was enriched by the resource consumption of the patterns by the aforementioned model-to-model transformation.

We developed transformations using QVT Operational language and Acceleo for model-to-model (M2M) transformation and for model-to-text (M2T) transformation, respectively. To traverse the model of the platform, the pattern system configuration and to perform calculations of resource consumption we developed a

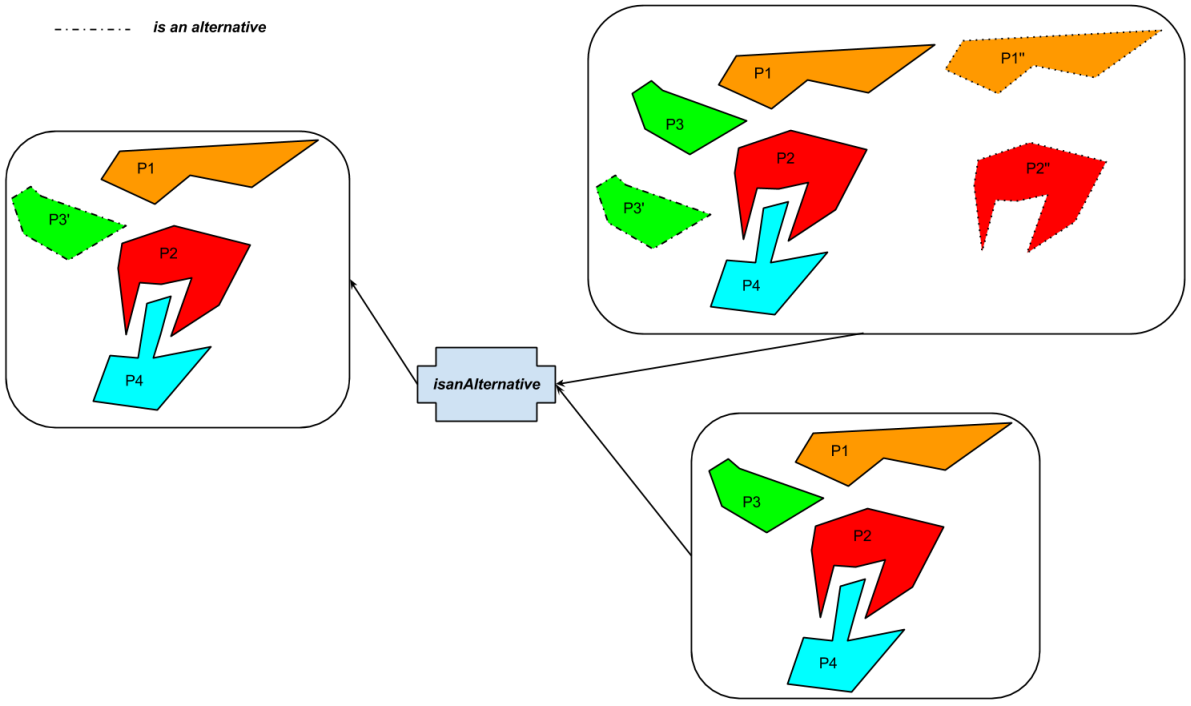

Figure 4.11: Generation of pattern system configuration- IsAnAlternative

M2M transformation. In addition, to enhance the readability of the analysis, we proposed to generate documentation using M2T transformation. The implementation of these two transformations is described in the following.

4.5.1 Calculating Resources Consumption - M2M

The model-to-model transformation, as depicted in the following Listing, calculates the resource usage of a pattern system configuration for a platform. The used metamodels (*CORE*, the common SEMCO metamodel, *GPRM* the properties metamodel, *SEPM*, the pattern metamodel and *SERM*, the resource metamodel) are declared (lines 1-4). The transformation is declared with input and output parameters typed by metamodels (line 6). The execution of the transformation is triggered by the main function (lines 8-10), calling the mapping *AnalyzeProperties* on each *SermProperty* of the `Platform`.

The mapping *AnalyzeProperties* (lines 12-18) parses the properties to calculate and fills in the used resources for the "computable" platform properties (line 13). The value of the attribute *valueSpecification* is calculated (line 15) by the helper *CalculateValue*.

For each computable property, the helper loops over all the patterns composing

the pattern system configuration (`PatternSystemConfiguration`). For each pattern (line 23), all its properties' categories (line 24) are compared the category of the `Platform` resource property, and if matching the values are summed up (line 30). The result (line 36) is then stored as the value of the *valueSpecification* of the "computable" platform resource property (line 15).

```
1
2  modeltype CORE uses "http://www.semcomdt.org/semco/core/2012/07";
3  modeltype GPRM uses "http://www.semcomdt.org/semco/tiqueo/gprm/2012/07";
4  modeltype SEPM uses "http://www.semcomdt.org/semco/arabion/sepm/2012/07";
5  modeltype SERM uses "http://www.semcomdt.org/semco/matho/serm/2012/07";
6
7  transformation CalculateResourceUsage(in PatternSystemConfiguration: SEPM, inout
       Platform: SERM);
8
9  main() {
10   Platform.objectsOfType(SermProperty) -> map AnalyzeProperties();
11 }
12
13 mapping inout SermProperty::AnalyzeProperties() {
14   if (self.computable) then {
15     if (self.valueSepcification.oclIsKindOf(GprmSingleValueSpecification)) then {
16       self.valueSepcification.oclAsType(GprmSingleValueSpecification).value := self
             .CalculateValue();
17       } endif;
18   } endif;
19 }
20
21 helper SermProperty::CalculateValue() : String {
22   var patterns := PatternSystemConfiguration.rootObjects()[SepmSystemOfPatterns].
         patterns;
23   var value : Real := 0.0;
24   patterns -> forEach(pattern) {
25     pattern.properties -> forEach(prop) {
26       if (self.category.=(prop.category)) then {
27         if (prop.valueSepcification.oclAsType(GprmSingleValueSpecification).value.
               oclIsUndefined()) then {
28           return 'ERROR in pattern '.concat(pattern.name);
29         }
30         else {
31           value := value.+(prop.valueSepcification.oclAsType(
                 GprmSingleValueSpecification).value.toReal());
32         } endif;
33
34       } endif;
35     };
36   };
37   return value.toString();
38 }
```

4.5.2 Documenting the Resources Consumption Analysis - M2T

The model-to-text transformation used for visualizing the results of the model to model transformation is based on Acceleo. The target is an HTML5 document showing the resource usage of the pattern system configuration by different resources, such as depicted in the following Listing.

The features of HTML page are augmented by the bootstrap[3] web front-end framework (e.g. line 12 for the css, lines 80-81 for the javascript). The page is structured with a menu for the different resources (lines 17-28) and a set of rows, one for each resource (lines 30-75).

The visual aids for determining if a resource's load exceeds the allowed limit, is a bar graph, which is colored following the traffic light rating system. Up to 75% of usage, it is green, from 75% to 100% it is amber and 100% or more it is red (lines 44-63).

```
1  [module generatePlatformDoc('http://www.irit.fr/semco/model/arabion/sepm/1.0.0', '
       http://www.semcomdt.org/semco/core/2012/07', 'http://www.semcomdt.org/semco/
       matho/serm/2012/07', 'http://www.semcomdt.org/semco/tiqueo/gprm/2012/07', 'http
       ://www.eclipse.org/emf/2002/Ecore')]
2
3  [template public generatePlatformDoc(aSermModel : SermModel)]
4  [comment @main/]
5  [file ('index.html', false, 'UTF-8')]
6
7  <!DOCTYPE html>
8  <html>
9  <head>
10 <title></title>
11 <meta name="viewport" content="width=device-width, initial-scale=1.0">
12 <link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
13 </head>
14 <body data-spy="scroll" data-target=".bs-docs-sidebar">
15
16 <div class="container">
17   <div class="row">
18     <div class="span3 bs-docs-sidebar">
19       <ul class="nav nav-list bs-docs-sidenav affix">
20         [for (resource : SermResource | aSermModel.resources)]
21         <li><a href="#[resource.name/]"><i class="icon-chevron-right"></i>[resource
               .name/]</a></li>
22         [/for]
23       </ul>
24     </div>
25     <div class="span9">
26       <div class="page-header">
```

[3] www.getbootstrap.com

```
27        <h1>[aSermModel.name/] <small>[aSermModel.description/]</small></h1>
28      </div>
29
30 [for (resource : SermResource | aSermModel.resources)]
31   <section id="[resource.name/]">
32   <h2>[resource.name/]</h2>
33   <div class="row">
34     <div class="span2"><h4>Name</h4></div>
35     <div class="span2"><h4>Max Value</h4></div>
36     <div class="span5"><h4>Usage in %</h4></div>
37   [for (property : SermProperty | resource.properties)]
38     [if (not property.computable)]
39       [for (propertyComp : SermProperty | resource.properties)]
40       [if (propertyComp.computable and property.category.name.matches(propertyComp.
             category.name))]
41       [let percentage : Real = (propertyComp.valueSepcification.oclAsType(
             GprmSingleValueSpecification).value.toReal() / property.
             valueSepcification.oclAsType(GprmSingleValueSpecification).value.toReal()
              * 10000).round()/100.0 ]
42         <div class="span2">[property.name/]</div>
43         <div class="span2">[property.valueSepcification.oclAsType(
               GprmSingleValueSpecification).value.toReal()/]</div>
44         <div class="span5">
45         <div class="row">
46         <div class="span1">[percentage/]</div>
47         <div class="span4">
48         [if (percentage >= 100.0)]
49           <div class="progress progress-danger">
50         [elseif (percentage > 75.0)]
51           <div class="progress progress-warning">
52         [else]
53           <div class="progress progress-success">
54         [/if]
55
56         [if (percentage >= 100.0)]
57           <div class="bar" style="width: 100%;"></div>
58         [else]
59           <div class="bar" style="width: [percentage.floor()/]%;"></div>
60         [/if]
61           </div>
62         </div>
63         </div>
64         </div>
65       [/let]
66       [/if]
67       [/for]
68     [/if]
69       </tr>
70   [/for]
71     <tbody>
72   </table>
73   </div>
74   </section>
75 [/for]
```

```
76
77   </div>
78
79 </div>
80 <script src="http://code.jquery.com/jquery.js"></script>
81 <script src="js/bootstrap.min.js"></script>
82 </body>
83 </html>
84
85 [/file]
86 [/template]
```

4.6 Conclusion

In this chapter, we have dealt with part 1 and part 2 of the ***RG1***. We have proposed languages for the specification of a set of modeling artifacts for the development of S&D pattern-based applications. These specification languages are based on metamodeling techniques that allow to specify property models, resources models and S&D pattern models.

The wanted role of pattern-use is to ease, systematize and standardize the approach to the construction of software-based systems. However, the problem consists in identifying them explicitly and then selecting them for reuse. This means that a careful balance and trade-off analysis among S&D requirements, patterns and available platform resources is necessary. In doing so, we propose to integrate a pattern in a system pattern configuration and to use model transformation techniques to calculate resource consumption values and to incorporate these information into system models.

In order to enforce reuse and to interconnect the process of the specification of modeling artifacts and the system development with these artifacts, we study in the next chapter a structured model-based repository of S&D patterns, property models and resource models. Therefore, instead of defining new modeling artifacts, that usually are time and efforts consuming as well as errors prone, the system developer merely needs to select appropriate patterns from the repository and integrate them in the system under development.

Chapter 5

A Model-based Repository

5.1 Introduction

In this chapter we present results related to the research goal 2 (***RG2***).

With regard to the definition of [9], here we go one step further: a model-based repository to support the specifications, the definitions and the packaging of a set of modeling artifacts to assist developers of trusted applications for resource-constrained embedded systems. Concretely, the repository system is a structure that stores specification languages, models and relationships among them, coupled with a set of tools to manage, visualize, export and instantiate these artifacts in order to use them in engineering processes (see Figure 5.1).

The remainder of the chapter is organized as follows. In Section 5.2 we give an overview of the proposed framework and identify the challenges for model-based repository coming from the TERESA project requirements and how they influenced the design decision made for our repository specification languages. In Section 5.3 we present the specification languages of the repository structure and its interfaces (APIs). In section 5.4 we illustrate the instantiation of the metamodel throw an example of the TERESA repository dedicated to S&D applications for RCES.

5.2 A Model-based Repository Framework

The repository presented here is a model-based repository of modeling artifacts. It constitutes one of the most important key elements in the engineering process for resource-constrained embedded systems. Now, we introduce our repository system framework tackling challenges that we present in the following.

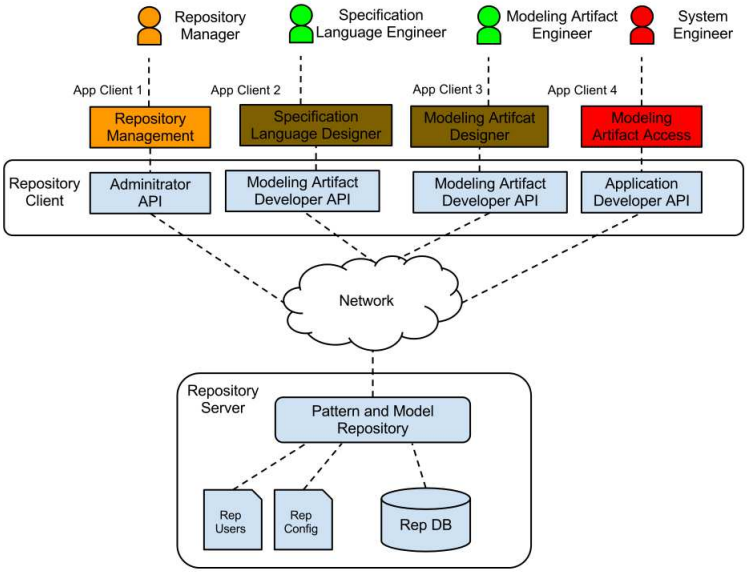

Figure 5.1: The repository system

Specification of the Repository Structure and Interfaces

The core of the framework is the definition of the specification languages to design the repository structure and the modeling artifacts, mainly S&D pattern, S&D property and resource property models. These languages are obtained by using metamodeling approaches as illustrated in the next sections. Here, we address the following challenge:

CHALLENGE 1: What kind of artifacts, how they will be organized and what kind of interfaces are needed for external tools?

This challenge may be declined on the the following concerns:

- Flexible structure: flexible structure supporting evolution (new kind of artifacts)
- Artifacts organization:
  - support specification languages and instances
  - system engineering lifecycle stage classification
  - support other end-user classification
- Interfaces specification:
  - extendable interfaces for multiple artifact design tools

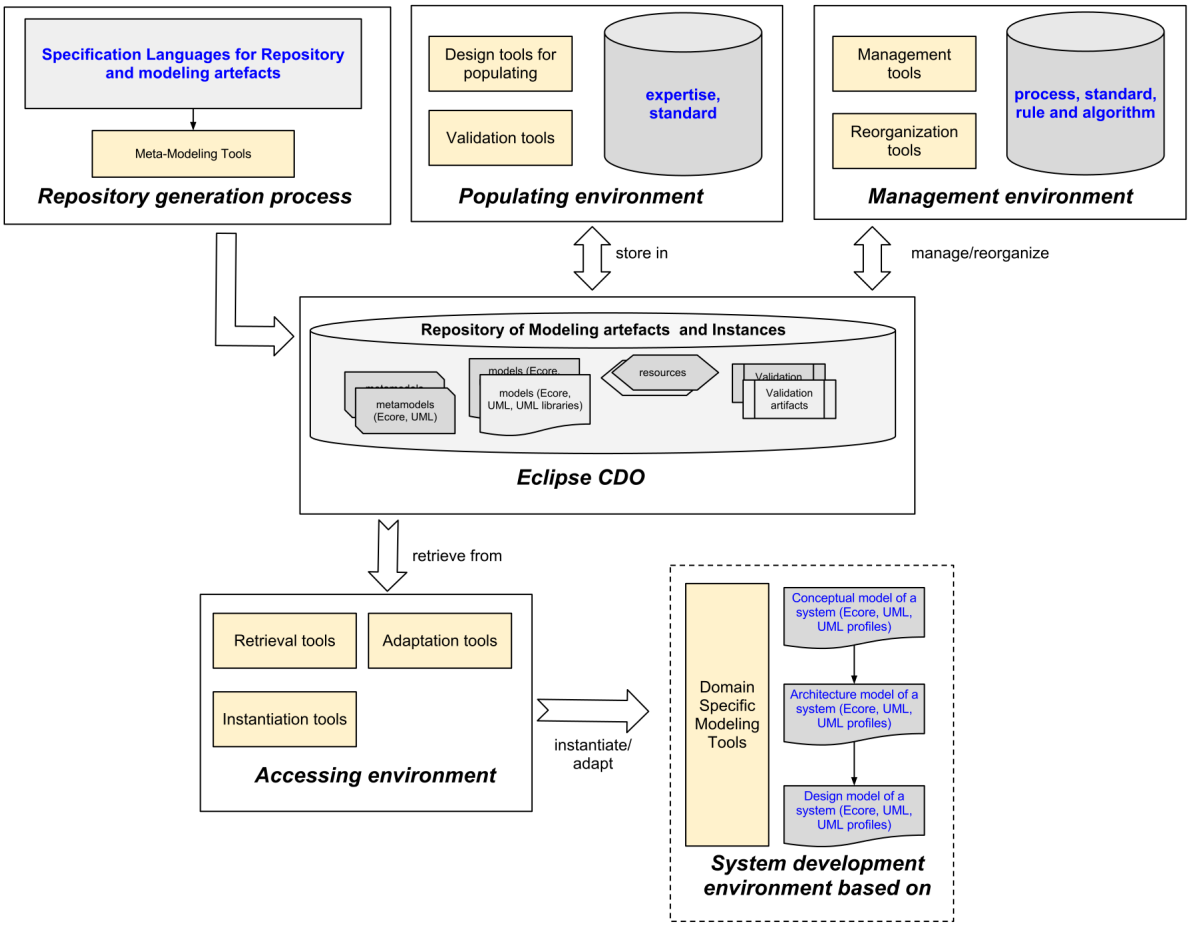

Figure 5.2: The proposed framework for the model-based repository system

– extendable interfaces for multiple system development IDEs

Implementing the Repository Infrastructure

Once these specification languages have been defined, it is possible to develop a repository in which the modeling artifacts and their related specification languages are stored. The development of such a repository is based on transformation techniques and the availability of MDE tools. As we shall see in Section 6.3, MDE transformation techniques are use for the automatic generation of the repository structure. This part targets:

CHALLENGE 2: What is the appropriate architecture style and what kind of technologies we will use ?

This challenge leads to the following concerns:

- Architecture and deployment style: client server, three-tiered architecture,...
- Components: storage, index, API, ...
- Supported format: XMI, UML, code, ...
- Technology used: Eclipse CDO, EMF Store, Java, ...
- External tools: Targets IDEs, design tools, ...

Populating the Repository

The development environment associated to the repository is based on the specification languages for defining modeling artifacts. It is composed of tools for the design, the instantiation and the validation of modeling artifacts. These tools support basic features including the storage in the repository. Here, we deal with:

CHALLENGE 3: Who among practitioners, researchers and industry is interested to contributed and how can we support them ?

CHALLENGE 3 deals with the following:

- Collection of artifacts
  - domain independent and domain specific S&D patterns
  - property models (S&D and resource)
  - validation models
  - domain independent and domain specific process models
- Sources
  - industrial applications domains
  - academic
  - standards
- Accompanying applications
  - design tools: domain specific design tools, UML design tool, ...
  - IDEs (Eclipse, ...)

Managing the Repository

In this part, we focus on the management of the repository content. We provide software to manage relationships among artifact model specifications and instances, and between artifacts and their related models. Moreover, we support basic features such as artifact management and user management. This part is related to:

CHALLENGE 4: How do we manage the repository content, the repository infrastructure, and the targets?

CHALLENGE 4 deals with the following:

- Artefact management
  - artefacts domain
  - reorganization of the repository to support specifying relationships across artefacts
  - indexing
- User management
  - user access control
  - user resources
- Domain management

Accessing the Repository

Here we focus on the repository accessing techniques providing simple interfaces one for each artifact kind (modeling language). By accessing the repository, some facilitators are provided guiding the end-user choices on modeling artifacts which can be used to satisfy the system-under-development's requirements. We will cover the following:

CHALLENGE 5: How the repository interact with domain specific development environment in different platforms ?

CHALLENGE 5 deals with the following:

- Accessing the repository through simple interfaces one for each artefact kind (modeling language)
- Search/browsing
  - keyword search
  - context search (application domain, ...)
- Artefacts retrieval
  - selection
  - instantiation
- Implemented as
  - Eclipse plugin application targeting Eclipse IDE
  - Standalone application targeting domain specific IDEs
  - Web-based access targeting web browsers

5.3 A Language for the Specification of the Repository

The specification of the structure of the repository is based on the organization of its content, mainly the modeling artifacts and the specification languages. Moreover, we identified an *API* as a specification of the repository interaction system architecture. That is, we propose a metamodel to capture these two main parts: the first one is dedicated to store and manage data in the form of *Compartments*, the second one is about the *Interfaces* in order to publish and to retrieve patterns and models and to manage interactions between users and the repository.

5.3.1 Repository Structure Metamodel

The principal classes of the metamodel are described with Ecore notations in Figure 5.3. The following part depicts in more details the meaning of principal concepts used to structure the repository:

- *SarmRepository.* Is the core element used to define a repository.

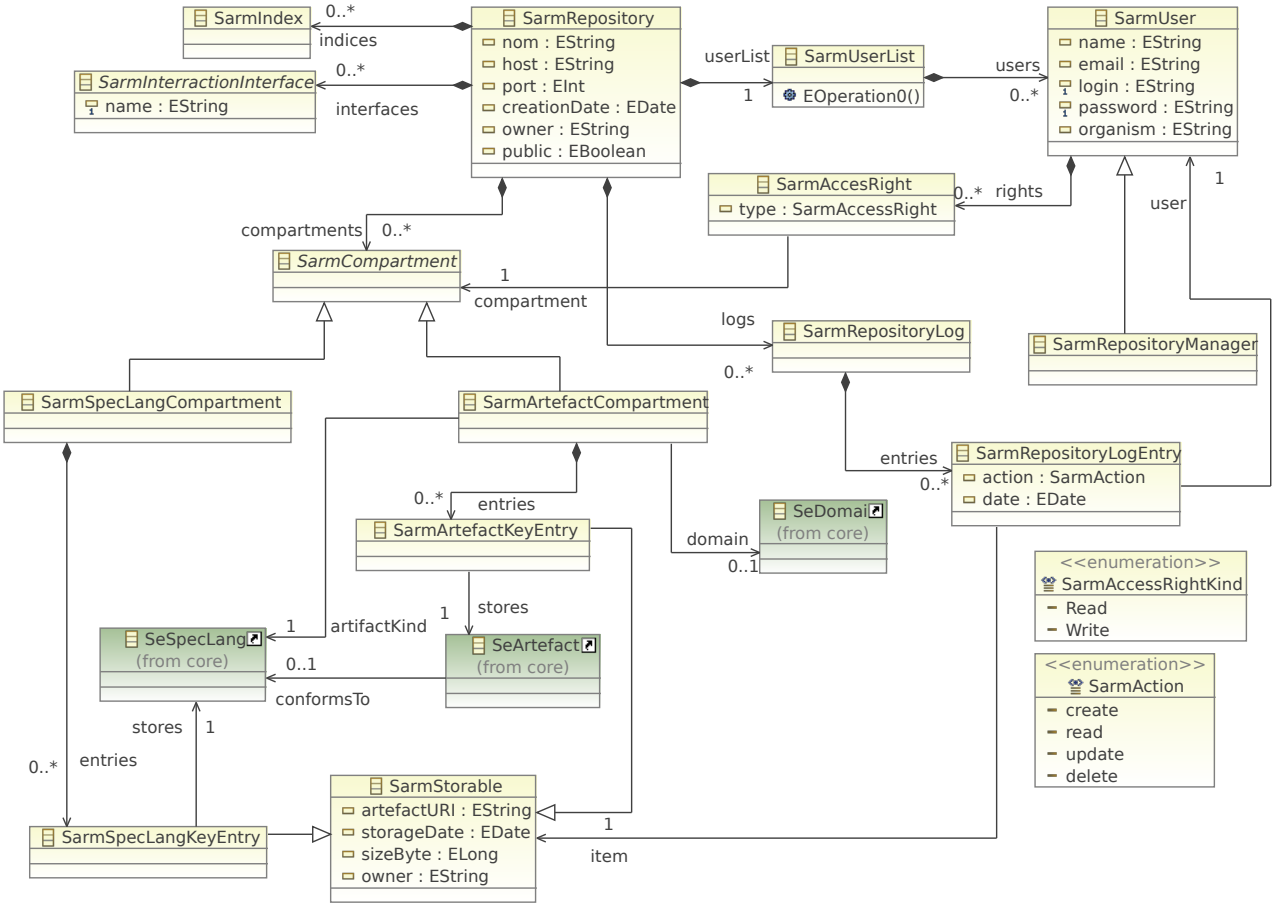

Figure 5.3: SARM - Structure

- *SarmCompartment.* Deals with the data and the data structure for processing the data. We have identified two principles kinds of compartments:
  - *SarmSpecLangCompartment.* Is used to store the specification languages of the modeling artifacts.
  - *SarmArtefactCompartment.* Is used to store the modeling artifacts specifications.
- *SarmStorable.* Is used to define a set of characteristic of the modeling artifacts, mainly those related to their storage. We can define: *artefactURI*, *storageDate*, *sizeByte*, etc...
- *SarmSpecLangKeyEntry.* Is the key entry to point towards a specification language model in the repository.
- *SarmArtefactKeyEntry.* Is the key entry to point towards a modeling artifact

specification in the repository.

- *SarmRepositoryLog.* Is used to store the repository log. The log will be used for repository access analysis in order to improve its structure and/or protection.
- *SarmAccesRight.* Is used to define the characteristics regarding the access right to the repository and its its content.
- *SarmUser.* Is used to define the user profile.
- *SarmUserList.* Is used to store the list of users in the repository.

5.3.2 Repository Interfaces Metamodel

Regarding interaction, the repository exposes its content through a set of interfaces, as depicted in Figure 5.4. The meaning of the proposed concepts are presented in the following:

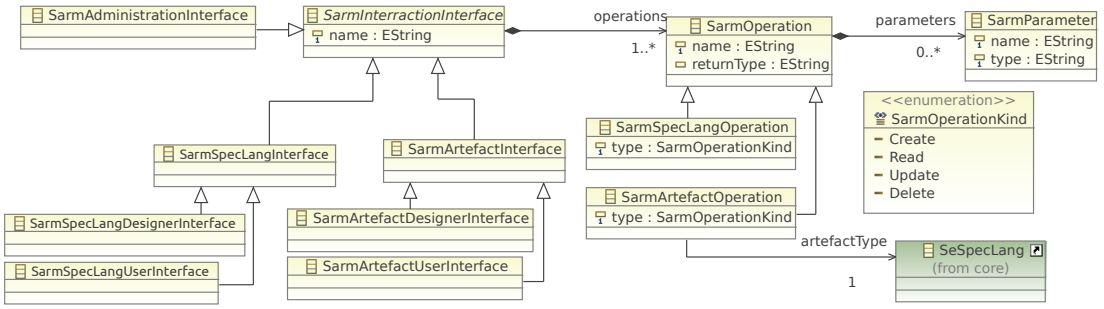

Figure 5.4: SARM - Interfaces

- *SarmInteractionInterface.* In addition to the repository structure, we proposed a specification of the interfaces (APIs) to exhibit the content of the repository and for repository management. The *API* exposes the following interfaces as shown in Figure 5.4:
  - *SarmAdministrationInterface.* Manages the repository.
  - *SarmSpecLangDesignerInterface.* Offers a set of operations including the connection/disconnection to the repository and to populate the repository with metamodels.
  - *SarmSpecLangUserInterface.* Offers a set of operations mainly connection/disconnection to the repository, search/selection of the specification languages.

- *SarmArtifactDesignerInterface.* Offers a set of operations including the connection/disconnection to the repository and to populate the repository with artifacts.
- *SarmArtefactUserInterface.* Offers a set of operations mainly connection/disconnection to the repository, search/selection of the modeling artifacts.

5.4 A Model-based Repository for S&D Applications in RCES

The main goal of the repository in our context is to share expertise, interacting with existing engineering process, in order to build S&D applications for resource-constrained embedded systems. As stated in the previous sections, we have identified three kinds of artifacts: *Patterns*, *Properties* and *Resources*.

5.4.1 Repository Structure Model

The approach was evaluated in the context of the TERESA project[1] for several application domains including Railway, Metrology, Home Control and Automotive. Here we focus on the Railway domain. We propose to specify a repository called *TeresaRepository* for the modeling artifacts and dedicated to Railway domain.

- *Repository.* We define *TeresaRepository* as an instance of the *SarmRepository*.
- *Compartment for languages.* we define *MetamodelCompartment* as an instance of the *SarmCompartment* to store the metamodels studied in chapter 4, mainly pattern, resource and property metamodels.
- *Compartment for artifact.* We define 11 compartments to store the artifacts:
  - *DIPCompartmentSystem.* Is used to store Domain Independent System Patterns.
  - *DIPCompartmentArchitecture.* Is used to store Domain Independent Architectural Patterns.
  - *DIPCompartmentDesign.* Is used to store Domain Independent Design Patterns.

[1] http://www.teresa-project.org/

- *DIPCompartmentImplementation.* Is used to store Domain Independent Implementation Patterns.
- *DSPCompartmentSystemRailway.* Is used to store Railway Domain Specific System Patterns.
- *DSPCompartmentArchitectureRailway.* Is used to store Railway Domain Specific Architectural Patterns.
- *DSPCompartmentDesignRailway.* Is used to store Railway Domain Specific Design Patterns.
- *DSPCompartmentImplementationRailway.* Is used to store Railway Domain Specific Implementation Patterns.
- *RPCCompartment.* Is used to store Resource Property libraries of units, types, categories.
- *SDPCCompartment.* Is used to store S&D Property libraries of units, types, categories.
- *RCCompartment.* Is used to store libraries of Resource categories .

An implementation as an Eclipse plugin using CDO is described in Section 6.3.

5.4.2 Repository Interfaces Model

In addition to the repository structure, we present in the following an example of a model of interfaces (APIs) to exhibit the content of the repository of S&D patterns and its management.

- Management. we define *AdministrationAPI* as an instance of *SarmAdministrationInterface* for the management of the repository. The interface provides the following operations:
  - addUser(): adds a user with login/password and access rights to compartments (UserCompartmentAccess),
  - createRepository(): creates a Repository on a certain machine, with Repository Name, using login and password as credentials,
  - deleteArtifactFromCompartment(): deletes an artifact from a compartment,
  - deleteUser(): deletes the user with login,

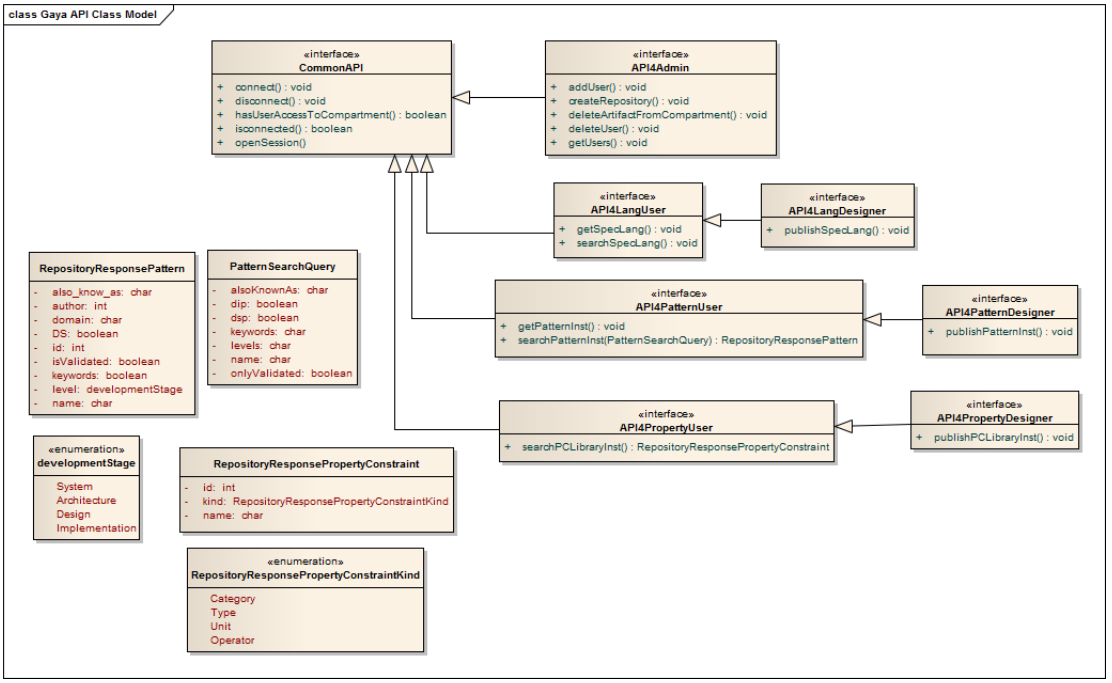

Figure 5.5: The Repository Interfaces and Classes

- emptyCompartment(): deletes all the entries from a compartment
- getUsers(): retrieves a list of all users.

In order to manipulate the modeling artifacts the following APIs are defined :

- *PatternDesignerAPI.* Is an instance of *SarmArtifactDesignerInterface* used to manage the lifecycle of pattern artifacts. The interface provides the following operations:
  - publishPatternInst(): publishes a Pattern to the right Compartment.
- *PatternUserAPI.* Is an instance of *SarmArtifactUserInterface* used to search/select pattern artifacts. The interface provides the following operations:
  - searchPatternInst(): searches for Patterns in the repository following the *query* given as parameter and returns a List of *RepositoryResponsePattern* which gives an overview of the patterns which fit the query,
  - getPatternInst(): gets the Pattern with ID uid from the repository and stores it as a file with filename.

- *PropertyDesignerAPI.* Is an instance of *SarmArtifactDesignerInterface* used to manage the lifecycle of property library artifacts, including unit, type and category libraries.
  - publishPCLibraryInst(): publishes a P&C Library to the right Compartment.
- *PropertyUserAPI.* Is an instance of *SarmArtifactUserInterface* used to search/select property library artifacts, including unit, type and category libraries. The interface provides the following operations:
  - searchPCLibraryInst(): searches for P&C Libraries in the repository following the *query*given as parameter and returns a List of *RepositoryResponsePropertyConstraint* which give an overview of the P&C Libraries which fit the query,
  - getLibraryInst(): gets the P&C Library with ID uid from the repository and stores as a file with filename.
- *ResourceDesignerAPI.* Is an instance of *SarmArtifactDesignerInterface* used to manage the lifecycle of resource category library artifacts.
  - publishRCategoryLibraryInst(): publishes a Resource category library to the right Compartment.
- *ResourceUserAPI.* Is an instance of *SarmArtifactUserInterface* used to search/select resource category library artifacts. The interface provides the following operations:
  - searchRCategoryLibraryInst(): searches for Resource category Libraries in the repository following the *query*given as parameter and returns a List of *RepositoryResponseResourceCategory* which give an overview of the Resource category Libraries which fit the query,
  - getRCategoryLibraryInst(): gets the Resource category Library with ID uid from the repository and stores as a file with filename.

An implementation of the APIs as Eclipse plugin is described in Section 6.3. Note that, the APIs are provided as open source software: extensible and customizable.

5.5 Transformation for the Instantiation

The resulted software repository tool needs to export patterns so they are understandable by the tools of the target process. This would reduce the disruption of workflow caused by the need to switch to another tool in order to apply the model-based repository approach in the pattern-based S&D application development process.

Because of the variety of tools used during the development of an application, such as requirement tracking software, modeling tools or IDEs, the tool also needs to know the current phase in order to adapt its output to the correct tool according to the domain process model. In the case of the integration with Rhapsody as described in Chapter Study of Railway Domain. The repository tool should be able, for example, to adjust the level of detail in the pattern according to the reported phase to reflect refinements.

5.5.1 Repository Instantiation into UML Modeling Environment - M2M

This work leverages the model-to-model transformation engine in order to ensure the translation of the pattern from its repository storage format (conform to the SEPM metamodel) to the expected target format (e.g. General Purpose Modeling and Domain Specific Languages). For instance, if the target is UML tool Rhapsody then the resulted transformation output is a subset of UML which can be imported by Rational Rhapsody.

In our context, we define a one-to-one mapping between the SEPM concepts and the UML component concepts as shown in Figure 5.6. The implementation of this transformation using QVT Operational language is presented in Section 5.5.2.

5.5.2 Implementation of Transformation

For the pattern's instantiation, from the repository to the developer's modeling environment, we developed M2M transformations. Table. 5.1 presents the mapping we propose.

Here we further clarify the principal pattern artifacts.

A *Pattern* is represented by a complex component (i.e. composite). The mapping between a Pattern and a Component is a logical choice because our pattern vision is already done on the component concept. The diagram of representation is the

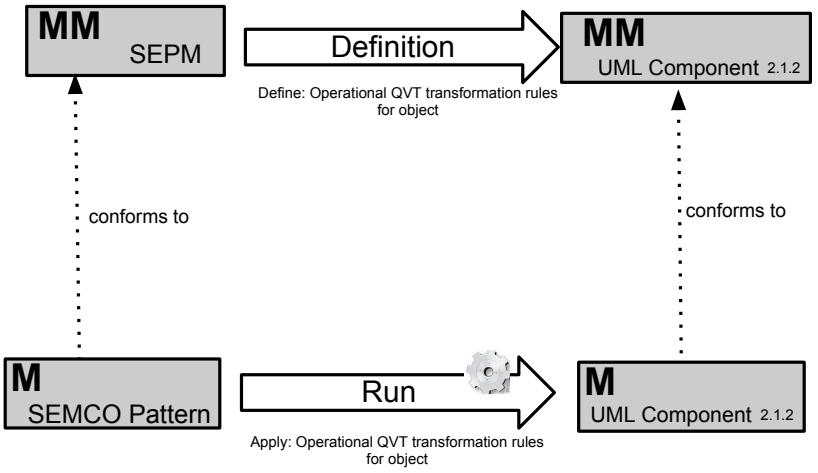

Figure 5.6: SEPM to UML Component Transformation

SEPM Concepts	**UML Component Concepts**
Pattern	Complex Component
Interfaces of a Pattern	Ports/Interfaces
Constraint	OCL Constraint
Property	DataType
Static Internal Structure of a Pattern	Internal Structure of a Complex Component
Dynamic Internal Structure of a Pattern	Sequence Diagram

Table 5.1: SEPM to UML Component Mapping

Component diagram / Composite structure. In the UML superstructure v2.1.2 the Basic Components package defines the concept of a component as a specialized class that has an external specification in the form of one or more provided and required interfaces, and an internal implementation consisting of one or more classifiers.

Interfaces of a pattern are represented by *Ports/Interfaces (provided/required).* In UML superstructure v2.1.2 a *Port* may specify the services a classifier provides (offers) to its environment as well as the services a classifier expects (requires) of its environment. We can use two ports: the first one to gather all the *external interfaces* and the second one to gather all the *internal (technical) interfaces.*

Properties of a pattern are represented by *DataTypes.* In UML superstructure v2.1.2 when a property is owned by a classifier other than an association via *ownedAttribute*, it represents an attribute of the class or data type.

The *Static Internal Structure* of a pattern is represented by the *Internal Structure* of a complex component (i.e. assembly of Basic Component). The diagram of representation is the Component diagram / Composite structure.

Figure 5.7 shows an overview of a set of transformation rules using QVT under EMF. SEPM and UML_COMPONENT[2] are specified using Ecore and act, as source and target metamodel for the transformation rules respectively.

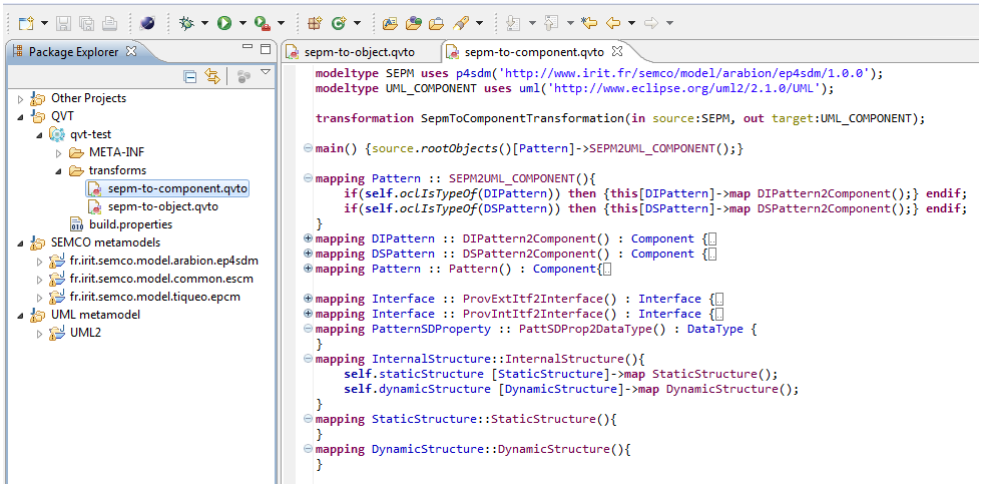

Figure 5.7: Mapping rules from SEPM concepts to UML Component Concepts using QVTO

5.6 Conclusion

To systemically model a repository of modeling artifacts for S&D applications, this chapter presents a specification language, which fulfill the first part of ***RG2*** that uses metamodeling and model transformation techniques. The resulting repository system, as a data structure storing artifacts and allowing 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, property and resource modeling languages used in this thesis have evolved compared to ones presented in the TERESA deliverables (D4.2 [71] and D4.4 [72]). 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 points out the major industrial requirements that the framework now meets. One of them is the repository storage and the support for interactions with the artifact and the system development lifecycles. A repository of S&D patterns allows reusing validated patterns. A pattern designer defines patterns and stores them

[2]The part of UML responsible for the representation of Component-Oriented Design

in the repository. A system designer reuses existing patterns from the repository through instantiation mechanisms which lead to simpler and almost seamless designs with quality improvements and cost savings. 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.

Chapter 6

Architecture and Implementation of Tools

6.1 Introduction

In this chapter, we propose an Model-Driven Engineering tool-chain supporting the PBSE (Pattern-Based System Engineering) methodology, and hence to assist the developers of secure and dependable systems. With regard to our contributions, here we deal with part 3 of ***RG1*** and part 2 of ***RG2***. The framework is centered around a model-based repository of S&D patterns and models providing an operational repository, tools for managing, tools for populating and tools for accessing the repository. At the core of the framework is a set of Domain Specific Modeling Languages (DSML)[22, 26, 34, 42] that allow modeling S&D patterns, property models, resource models and the repository structure. Using the proposed metamodels and the Eclipse Modeling Framework (EMF) [67], ongoing experimental work is done with SEMCOMDT[1] (SEMCO Model Development Tools, IRIT's editor and platform plugins).

The rest of this chapter is organized as follows. We start with the presentation of the architecture and the functionalities of the tool-chain. Then, we present in detail the implementation of the tools composing this tool-chain. We present the repository implementation, 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 and accessing the repository for reuse. Further, we discuss how to enable a set of transformations features, presented in Section 4.5

[1] http://www.semcomdt.org

and Section 5.5, in the editors. In the domain of assistance to the developement of S&D-based applications for RCES, we developed an editor for the specification of the pattern system.

6.2 Implementation Architecture

To tackle secure and dependable system engineering challenges, in the context of the PBSE methodology, we are developing an integrated set of software tools to enable S&D embedded system applications development by design. These tools improve the design, implementation, configuration and deployment of S&D RCES applications. In fact, capturing and providing this expertise by means of a repository of S&D patterns can support and improve embedded systems development. The following details this software system from the installation, over modeling artifacts development and reuse, evolution and maintenance for acquiring organizations, end-users and front-end support provider.

6.2.1 Tool-suite Architecture

SEMCOMDT provides three integrated sets of software tools: (i) *Tool set A* for populating the repository, (ii) *Tool set B* for retrieval and transform from the repository and (iii) *Tool set C* for managing the repository. As shown in Figure 6.1, thanks to UML component diagram the tool-suite is composed of:

- *Gaya (G).* for the repository structure and interfaces conforming to *SARM*,
- *Tiqueo (T).* for specifying models of S&D properties conforming to *GPRM*,
- *Arabion (A).* for specifying patterns conforming to *SEPM*,
- *GayaAdministrator (Admin).* for the repository management,
- *AccesTool (Access).* for the repository access.

In addition, we provide *Matho (M).*, a design tool for specifying resources conforming to *SERM*. The functionality and the detail of the implementation of the resource designer *Matho* are similar to the *Arabion* tool.

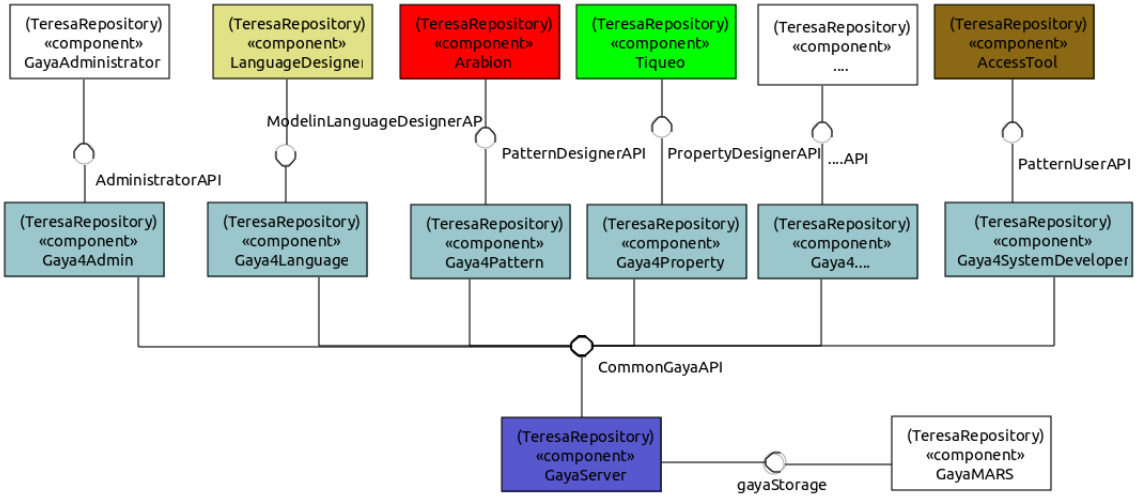

Figure 6.1: The tool suite architecture

6.2.2 Tool-suite Functionalities

In this section we present the design tools proposed for the repository populating, repository management and the repository accessing.

Repository Setup. GAYA is a repository platform to store the modeling artifact specifications and instances through the APIs. The server part is responsible for managing and storing the data, and provides a set of features to interact with the repository content. As shown in Figure 6.1, the server part is composed of two components: (1) *GayaServer* providing the implementation of the common API and (2) *GayaMARS* providing the storage mechanisms. The client part is responsible for populating the repository and for using its content-providing APIs interfaces for applications, such as depicted in Figure 6.1, in order to populate, access and to manage the repository. For instance, *Gaya4Pattern* (implements the API4PatternDesigner), *Gaya4Property* (implements the API4PropDesigner), *Gaya4Resource* (implements the API4resourceDesigner), *Gaya4Admin* (implements API4Admin) and *Gaya4SystemDeveloper* (implements the API4PatternUser).

Repository Populating - Design Tools. The property designer (Tiqueo), to be used by a *property designer*, provides features for specifying models of properties. In addition, Tiqueo provides some features to create a library for reusable objects, like the types and units which allows property designer to use the libraries in a domain independent manner. Furthermore, Tiqueo includes mechanisms to validate

the conformity of the property libraries with GPRM metamodels and to publish the results to the Gaya repository using the *Gaya4Property* API.

The pattern designer (Arabion), to be used by a *pattern designer* provides a set of features for specifying domain independent and domain specific patterns. In addition, Arabion includes mechanisms to validate the conformity of the pattern with SEPM metamodel, the generation of documentation and to publish the results to the repository with the repository interfaces (*Gaya4Pattern* API).

Repository Management. For the repository management, to be used by *repository manager*, we provide a set of facilities for the repository organization allowing the enhancement of its usage using the *Gaya4Admin* API. We provide also basic features such as user, domain and artifact management. Moreover, we provide features to support the management of the relationships among artifacts specifications and between artifacts specifications and their complementary models.

Repository Accessing. For accessing the repository, to be used by a *system engineer*, the tool provides a set of facilities to help selecting appropriate patterns including *keyword* search, *lifecycle stage* search, domain independent vs. domain specific search and property categories search. The results are displayed in search result tree as System, Architecture, Design and Implementation patterns. The Tool includes features for exportation and instantiation as dialogues targeting domain specific development environment. Moreover, the tool includes dependency checking mechanisms. For example, a pattern can't be instantiated when a property library is missing: an error message will be thrown.

6.3 CDO Repository Implementation: Gaya

Our approach, relying on an MDE-based techniques to build a set of DSLs and thus in our context supporting automated model-based repository building, such as visualized in Figure 6.3. We provide the environment for the use of the resulted repository through APIs.

6.3.1 CDO Repository Implementation Architecture

We used the Eclipse EMF/CDO based Ecore technology architecture to create our repository system as shown in Figure 6.2. 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 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.

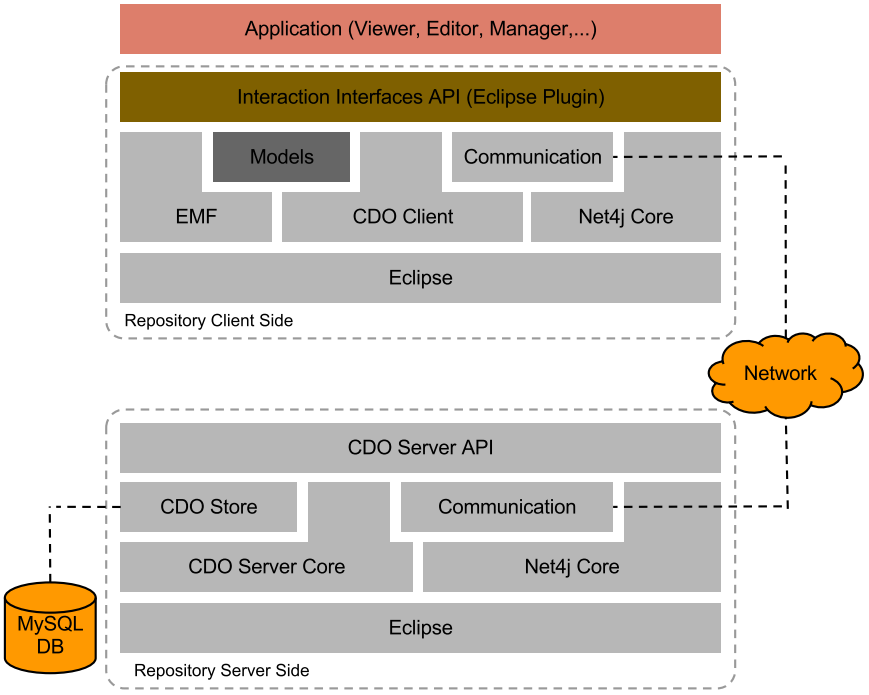

Figure 6.2: Repository implementation based on CDO

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 UML Components in Figure 6.1 (red components), 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 6.1. 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.

6.3.2 Repository Implementation Details

The models specifying the structure of the repository and the APIs are built through an EMF tree-based editor implementing the SARM metamodel, as shown in the top part of Figure 6.3. The resulting Ecore model is then used as input for the model-to-text transformations in order to generate the repository and APIs software implementation artifacts targeting the CDO platform [1] (as seen in the middle part of Figure 6.3).

The structure of the repository is derived from the repository structure model, proposed in Section 5.4, and implemented using Java and the Eclipse CDO Server technology. 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. The repository client APIs are derived from the repository APIs model, presented in Section 5.4, and then implemented as CDO clients. In our example, the repository interface models are visualized in Figure 5.5. We specified a set of functions and the data structure of their parameters in the form of UML class diagram. The implementation is based on the automatic code generation from the APIs model. In our development environment, the generated Java code defines the different interfaces and functions provided by the repository APIs. The skeleton of the API's implementations are then completed manually based on CDO technology. As the CDO server, the CDO clients are provided as Eclipse plugins (as shown in the lower part of Figure 6.3).

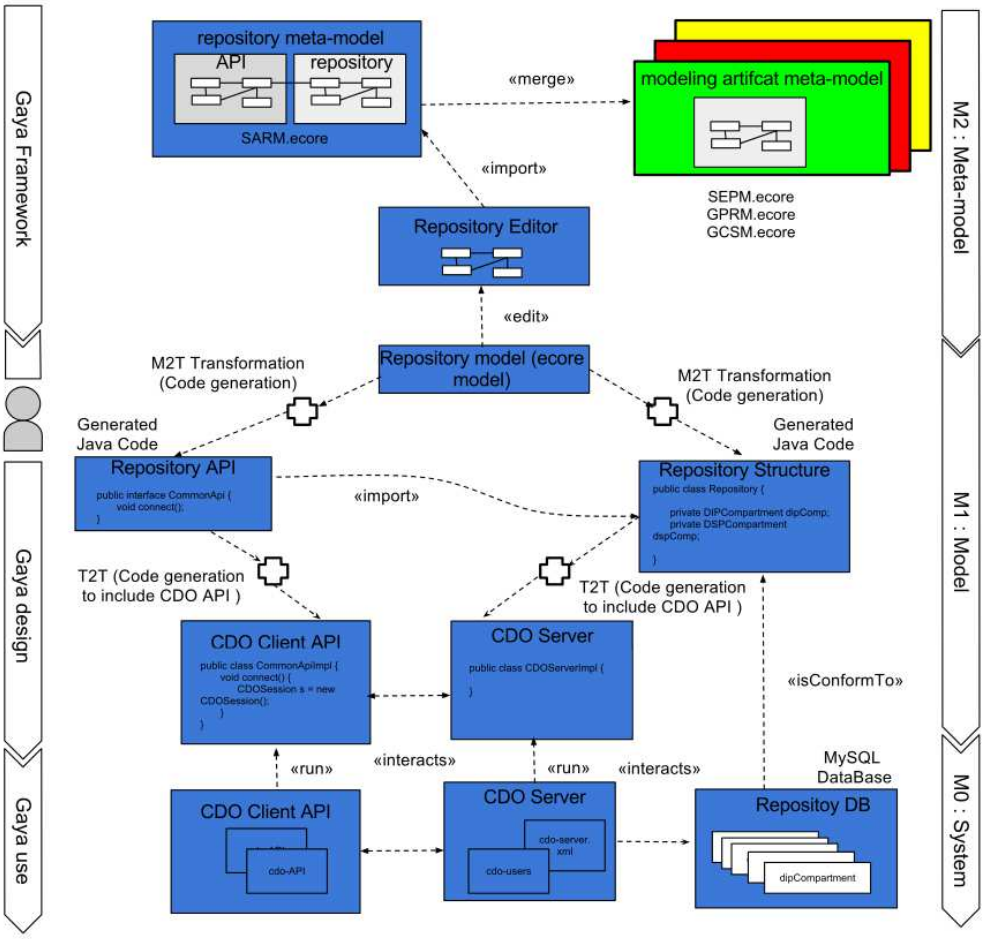

Figure 6.3: The Model-based repository building process

6.4 Design Tools for Repository Populating

As shown in Figure 2.6, we have used the Eclipse Modeling Framework (EMF) to support such a process and to develop our tool-suite: (1) we have defined our meta-models (MM) using the Eclipse Ecore and then (2) we have used the code generation techniques (EMF Generator Model - genmodel) provided by EMF to build our DSL editors. For each DSL, using the corresponding generator model we have generated the model, the edit and the editor code source. The generated editor code was modified to limit the user actions on the ones needed and to enhance user experience (e.g. modifying the name of some concepts). The tool provides facilities for editing modeling artifacts instances in a domain independent manner (DIM). Then the user can refine the guidelines for domain-specific application (DSM). Further, using EMF features, we added the metamodel conformance-checking functionality to the editor

(Static Model Checker) and code/documentation generation (Generator).

The second part of the design tools implementation was to create the HTML code generator based on Acceleo, a M2T component of the Eclipse Modeling Framework. We created two plugins, one for the HTML code generation and another as a user-interface plugin. The code generation plugin is based on the metamodel (Ecore file) to parse the model. We developed modularized code transformation templates, with every module template generating one HTML file per selected model element, and managing the links among them.

In the following we describe how the DSL process shown in Figure 2.5 is applied to build the property designer, resource designer and the pattern designer.

6.4.1 Property Modeling : Tiqueo

The Tiqueo tool is provided as an Eclipse Plugin, based on the Eclipse Modeling Framework Technology (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[2]. To create a property model, *Tiqueo* implements several facilities conforming to the GPRM metamodel to manage property libraries including units, types and categories. Tiqueo supports a number of features such as the modeling of a property library, validation, deposit and retrieval to and from the repository.

Tiqueo. The Tiqueo design environment is presented in Figure 6.4. There is a design palette on the right[3], a tree view of the project on the left and the main design view in the middle. Category library is built using type library instances. In our example, an instance of the *sdTypeLibrary* called *sdTypeLibrary.tm* is imported from the repository to the workspace using the *Retrieve* tool (see Section 6.5.1). Then, the user has to create a reference to this library as a resource, such as illustrated in Figure 6.5.

These libraries are used as external model libraries to type the properties of patterns and resources. Especially during the editing of the pattern (see Section 6.4.3) we define the properties using these libraries.

Property Library Validation. The property validation tool is used to guarantee design validity conforming to the property metamodel as visualized in Figure 6.6.

[2] http://www.semcomdt.org/semco/tools/updates/1.2
[3] (1) for unit, (2) for type and (3) for category.

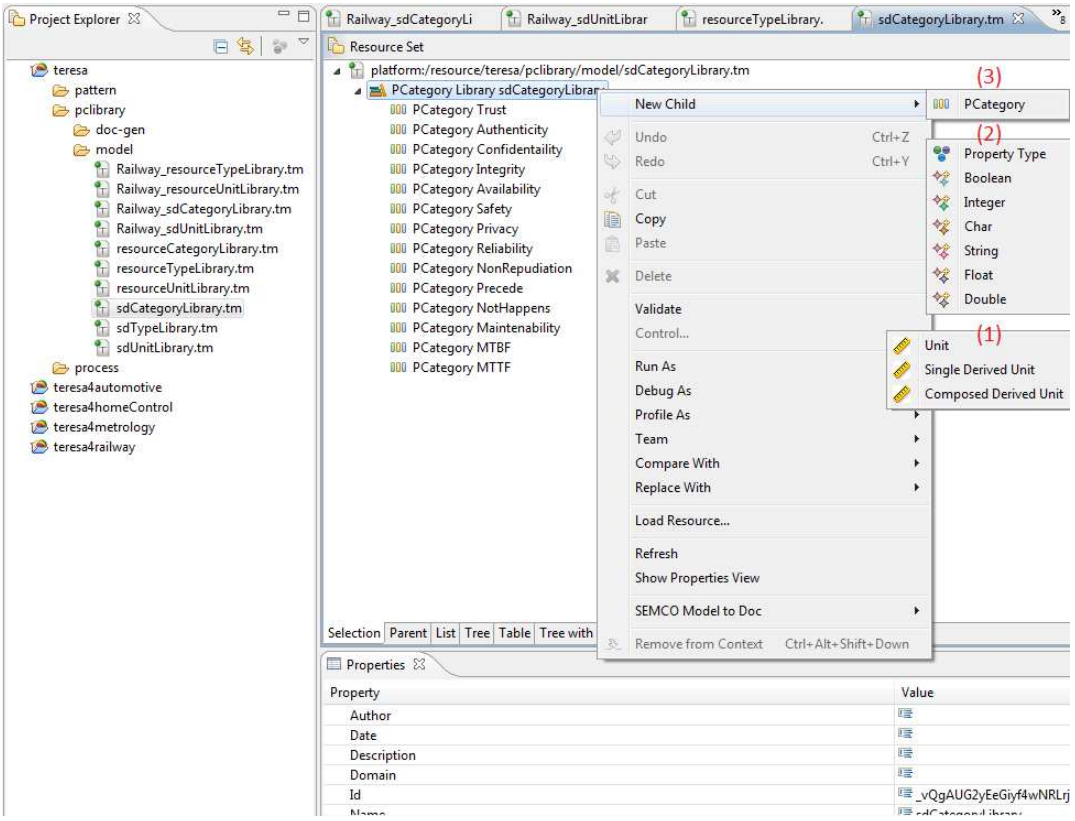

Figure 6.4: Designing a Category Library

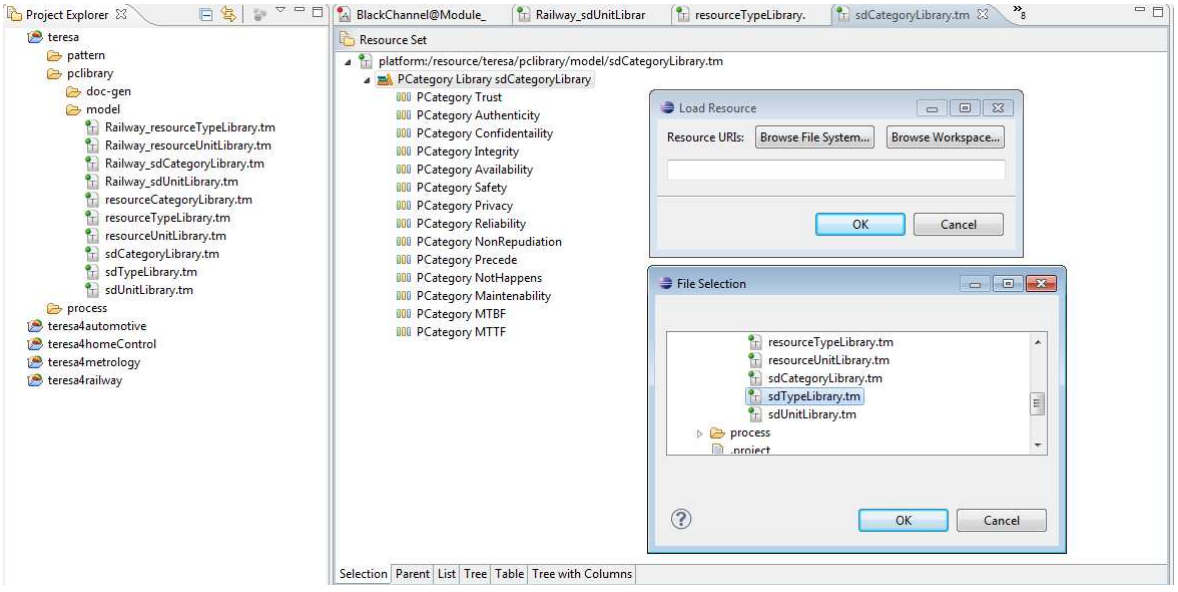

Figure 6.5: Eclipse Load Resource Tool

Property model validation starts by right clicking on the property model and selecting the *Validation* tool.

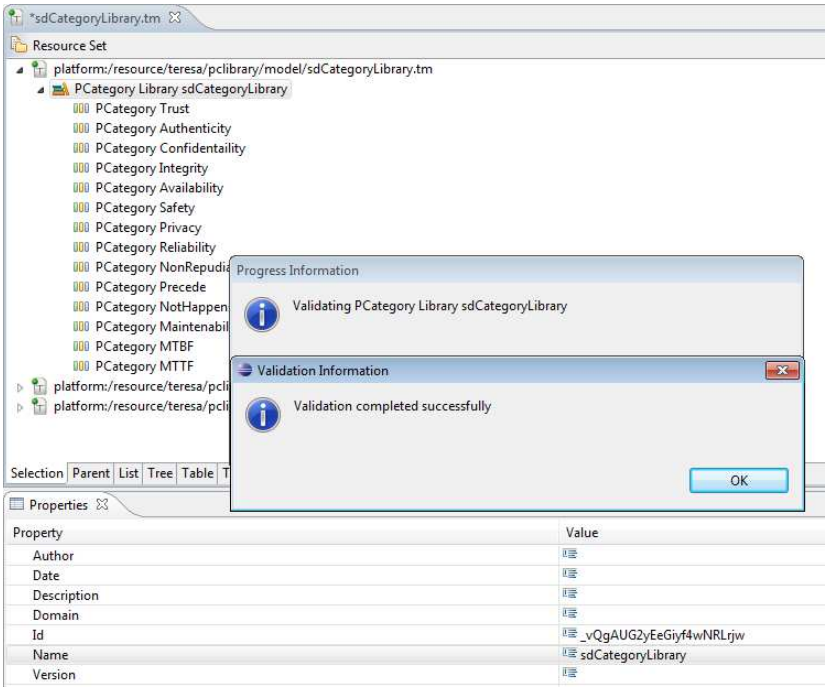

Figure 6.6: Property Library Validation

Property Library Deposit. Property library publication is triggered by running the *Publication* tool by right clicking on property model and selecting the *Publication* tool. The deposit tool requires the execution of the validation tool to guarantee design validity. When executed, as shown in Figure 6.7, the library will be stored in the repository. The tool uses the *Gaya4Property* API (see Section 5.4.2) for the deposit into the repository.

6.4.2 Resources Modeling: Matho

The resource designer called *Matho* includes features supporting the design of resource category libraries, the design of platform resources, validation, interaction with a verification framework, deposit to and retrieval from the repository. The Matho tool is provided as an Eclipse Plugin, based on the Eclipse Modeling Framework Technologies (EMFT).

Matho. The Matho design environment is presented in Figure 6.8. It contains a tree view of the project on the left and the main design view in the middle. When right clicking on the root element in the model, the design palette will appear allowing to add a resource category and/or a resource to model. Although the tool

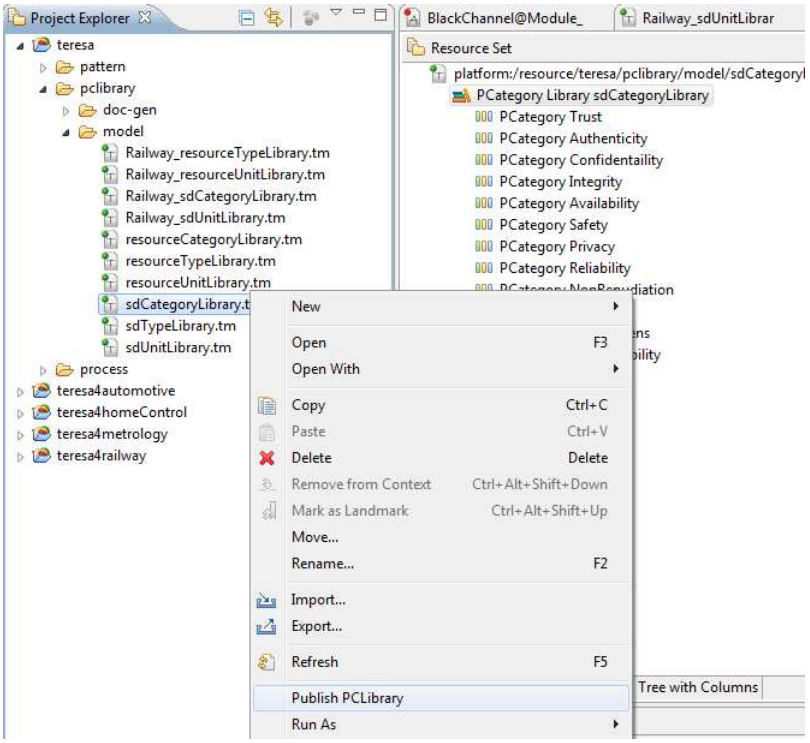

Figure 6.7: Property Library Deposit

allow to mix resource categories and resources in the same model (conforming to the definition of SERM metamodel), we recommend to separate these artifacts in two different models. The first, is used to define the libraries of resource categories; in our example the model is called *resourceCategoryLib.mm*; this is to foster the reuse of the same library with different resource models. The second, is used to describe the platform resources by importing and using the previous resource category libraries to type the resources.

Resource Library Validation. The resource validation tool is used to guarantee design validity conforming with resource metamodel as visualized in Figure 6.11. Resource model validation starts by right clicking on Resource model and selecting the *Validation* tool.

Resource Library Deposit. Resource publication is triggered by running the *Publication* tool. The tool starts by right clicking on resource model and selecting the *Publication* tool. When executed, the resource model will be stored in the repository following the resource designer's profile (compartment). The tool uses the *Gaya4Resource* (see Section 5.4.2) for publishing to the repository. Note, however

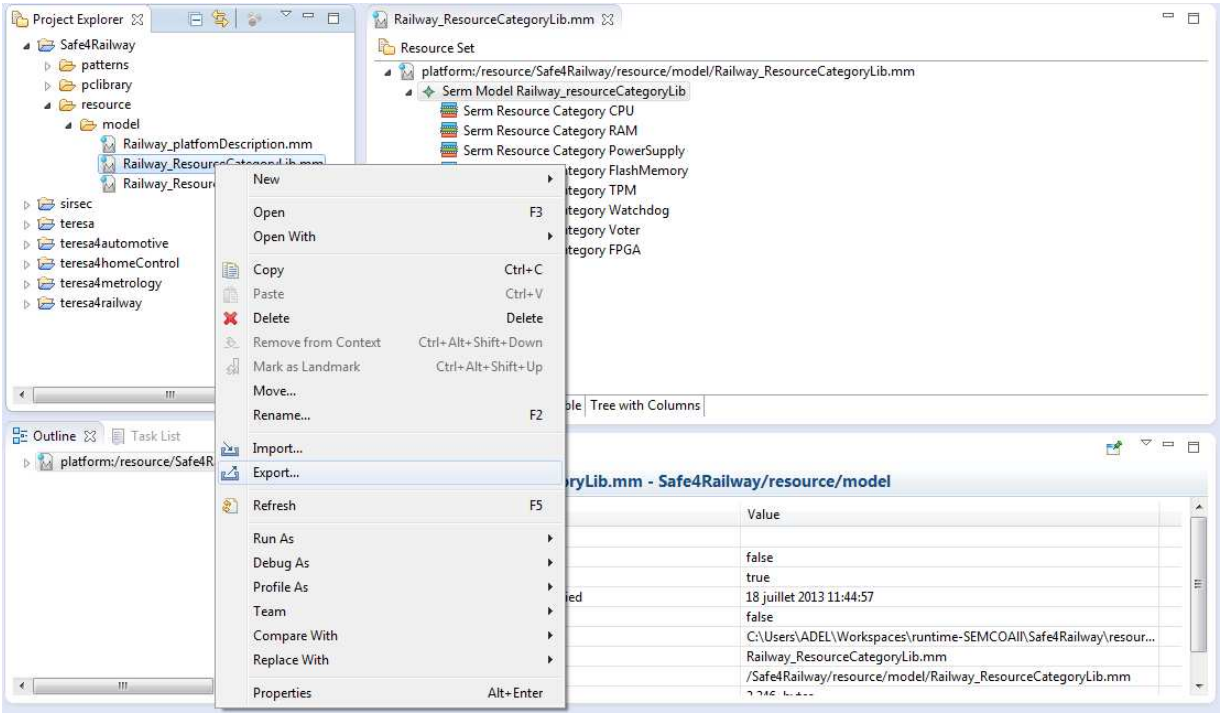

Figure 6.8: Matho Design Environment

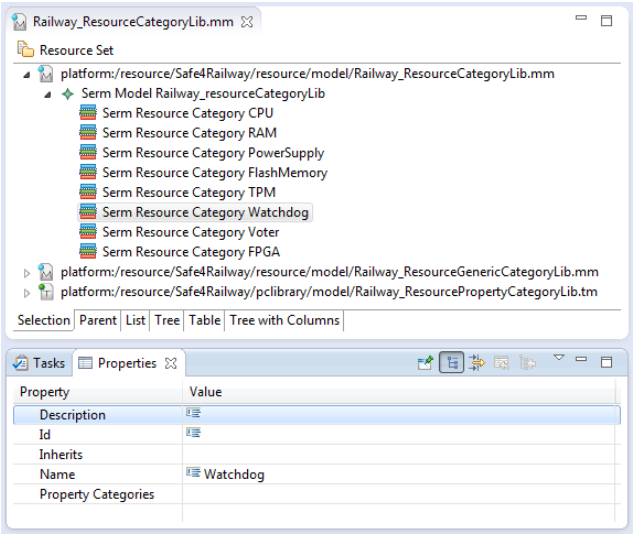

Figure 6.9: Designing a Resource Category Library

that the deposit tool requires the execution of the validation tool to guarantee design validity.

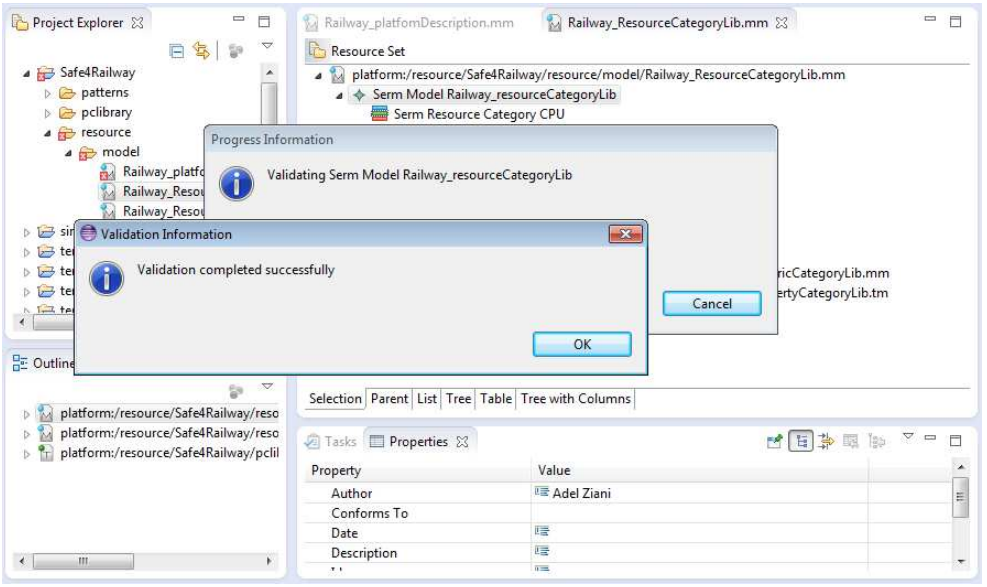

Figure 6.10: Designing the Platform Resources

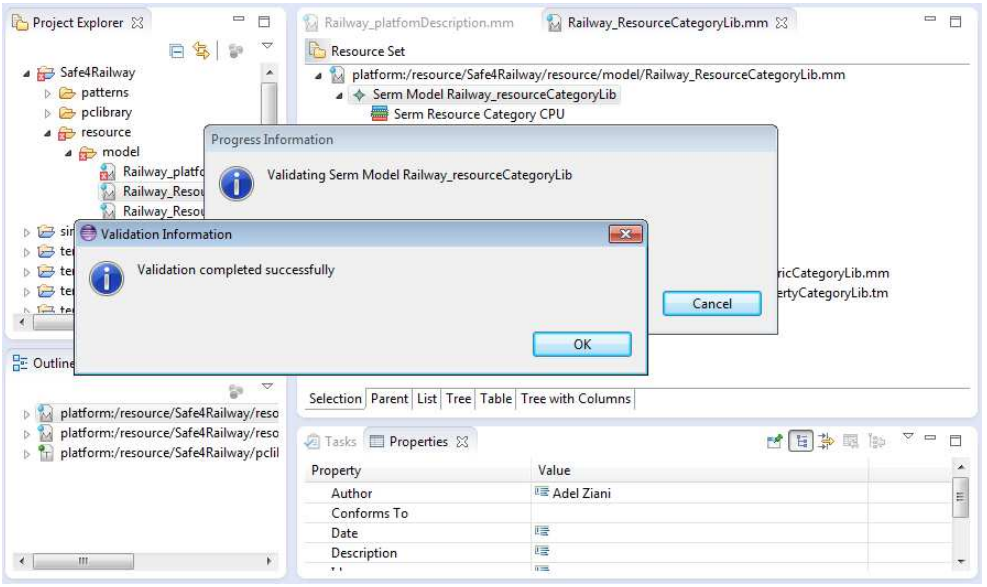

Figure 6.11: Resources Validation

6.4.3 S&D Pattern Modeling: Arabion

The pattern designer called *Arabion* supports a number of features including pattern design at DI (Domain Independent) and DS (Domain Specific) level, validation, interaction with a verification framework, deposit and retrieval into and from the repository, respectively. The Arabion tool is provided as an Eclipse Plugin, based on the Eclipse Modeling Framework Technologies (EMFT). We provide an installa-

tion 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[4].

Arabion. For an DI pattern, the design environment is presented in Figure 6.12. 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 internal structure was specified using UML diagrams created 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 the 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.

Pattern Model 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 model and selecting 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 6.13).

Patterns Model Deposit. Pattern publication is triggered by running the *Publication* tool, as visualized in Figure 6.14. The tool starts by right clicking on pattern model and selecting 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* for publishing to the repository. Note, however that the deposit tool requires the execution of the validation tool to guarantee design validity.

[4] http://www.semcomdt.org/semco/tools/updates/1.2

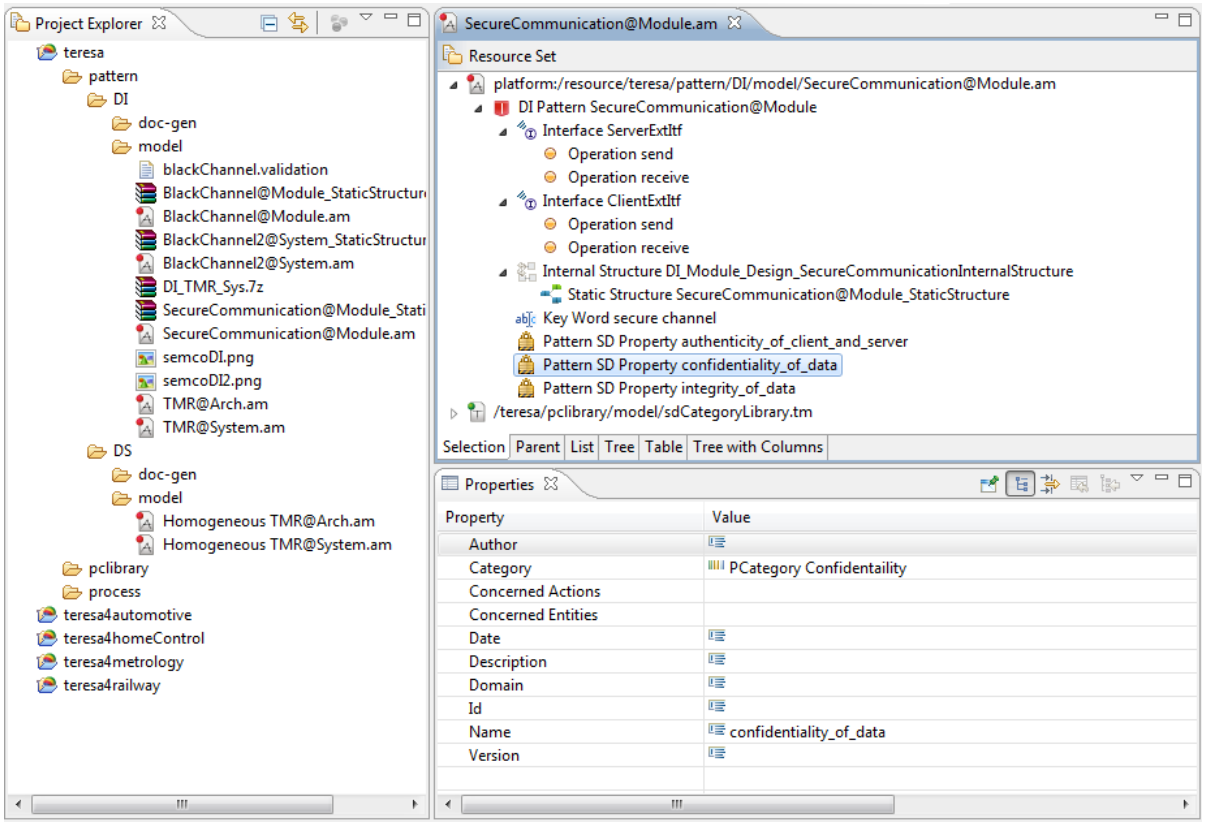

Figure 6.12: Secure Communication DI Pattern at Design level

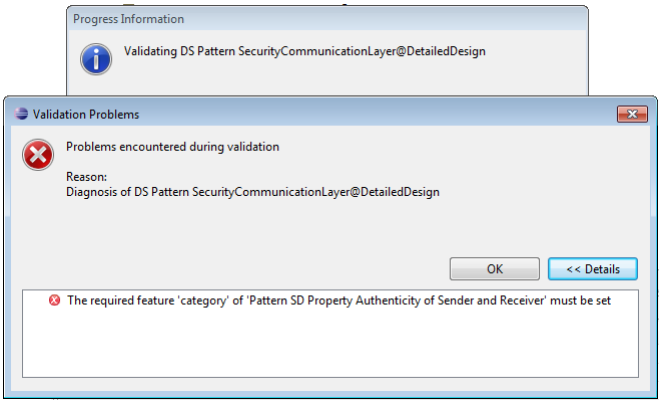

Figure 6.13: Pattern Validation

6.5 Repository Access-Tools

In this part, we present the repository tools for the assistance of the system development process. The access-tool supports the modeling artifact instantiation from the repository. By definition (see Section 2.7.1), the access-tool provides the following set of features:

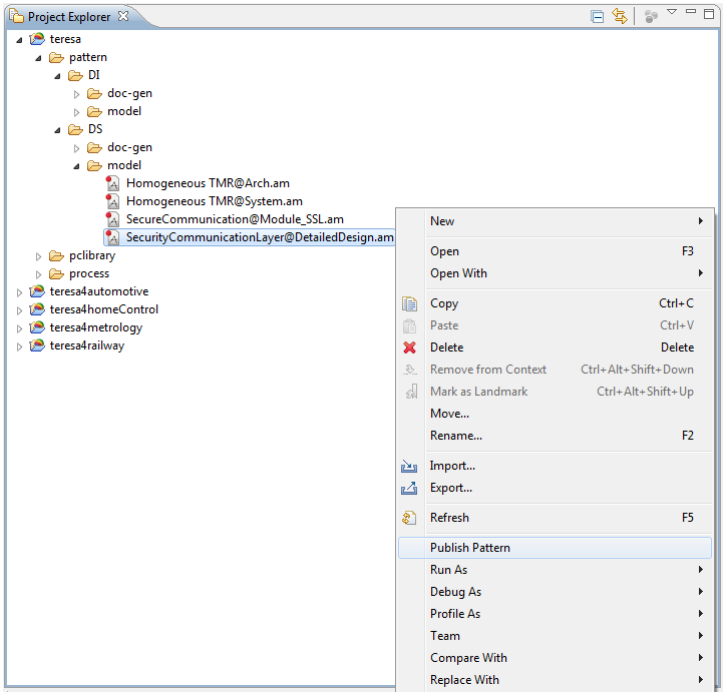

Figure 6.14: Pattern Publication

1. Support the definition of needs in terms of properties and/or keywords,
2. Support the search of modeling artifacts in the repository,
3. Support the selection of the appropriate modeling artifact from those proposed by the repository,
4. Support the adaptation of the selection into the development environment using model transformation techniques.

In the following, we refer to the first three features as Retrieval and the last feature as adaptation. In our case, the retrieval steps are applied to each of the modeling artifacts composing the repository and the adaptation steps are only related to patterns.

6.5.1 Retrieval

Property Library Retrieval. As mentioned before, when building a pattern we use property libraries to type its properties. The tool provides dialogues for selecting and instantiating libraries. The library search/selection dialogue is shown in the right part of Figure 6.15. 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 result tree as Unit Library, Type Library and Category Library. For example, the right part of Figures 6.15 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 6.15.

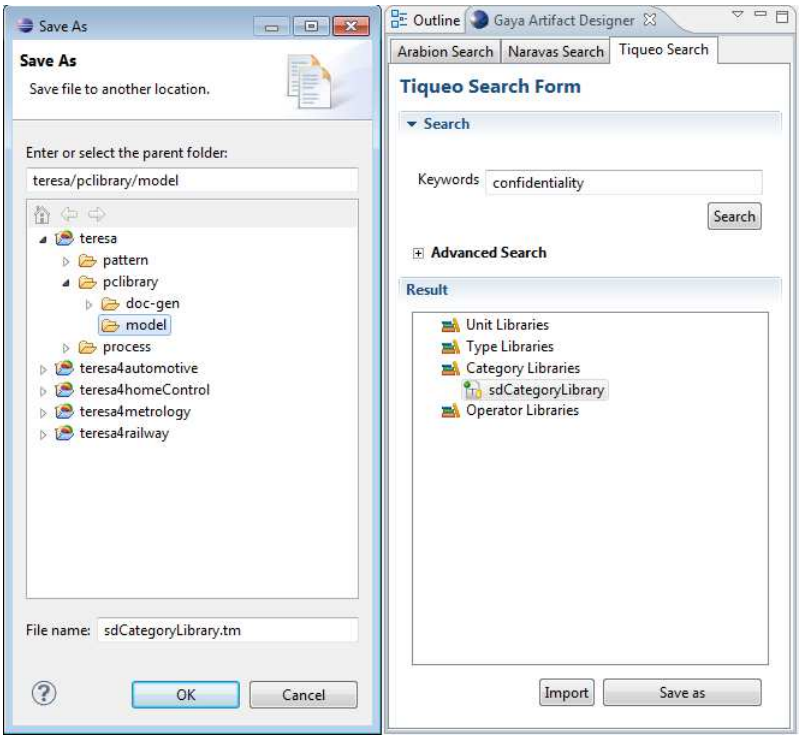

Figure 6.15: Property Library Instantiation

Pattern Model Retrieval. As mentioned before, DS patterns are built from DI patterns. The tool uses the *Gaya4Pattern* API for the search/selection of the patterns which is used during the pattern and the system development process. For instance, as shown in the right part of Figure 6.16, 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 6.16 shows that there is a DI pattern at design level targeting the *Confidentiality* S&D property[5], 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 6.16). 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 6.17).

6.5.2 Artifact Adaptation

In this thesis, we focus on the adaptation of pattern model. We developed in Section 5.5.2 an implementation of a transformation allowing to adapt an SEPM pattern to the UML Component using QVT Operational language.

6.6 Repository Administration

Repository management is implemented via the *GayaAdmin* tool. *GayaAdmin* offers repository management with facilities such as user management, artifact management and system of artifacts management. We offer these facilities through a set of dialogues triggered in the *GayaAdmin* tool. The main dialogue is shown in Figure 6.18.

6.6.1 User Management

The user management supports a set of features such as user lookup, add, remove, sorting and categorization as shown in the left part of Figure 6.19. The right

[5]In our modeling, this means that the pattern has a property with a confidentiality category type.

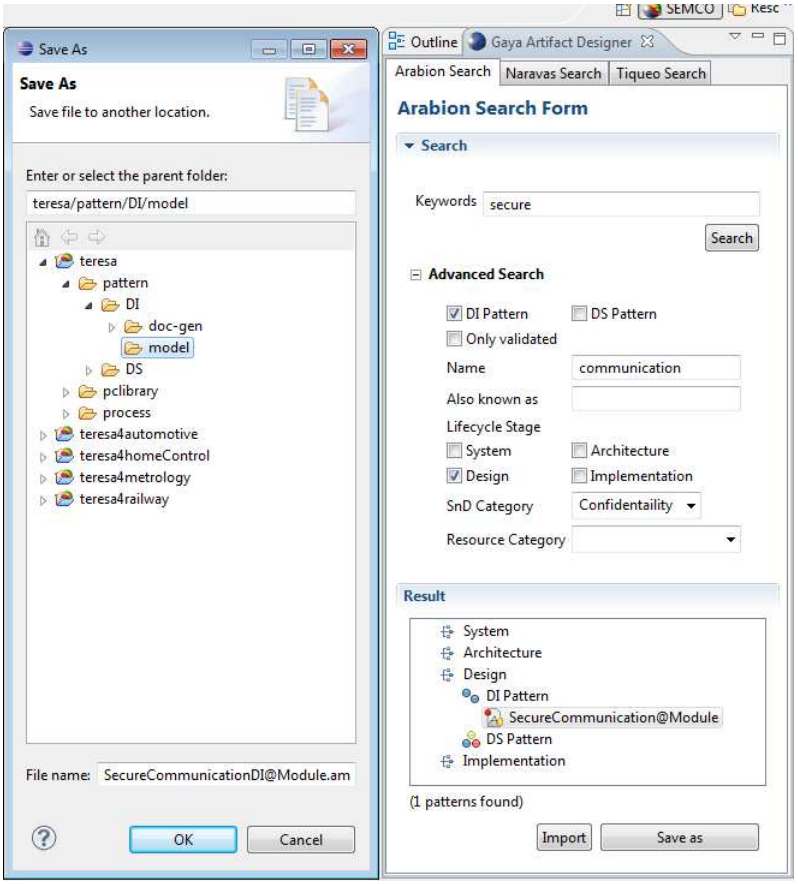

Figure 6.16: Pattern Instantiation

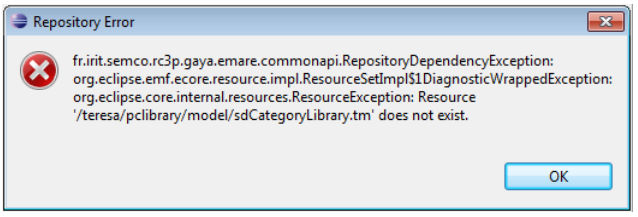

Figure 6.17: Pattern Instantiation - Consistency

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.

The user authentication dialogue is visualized in Figure 6.20. 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.

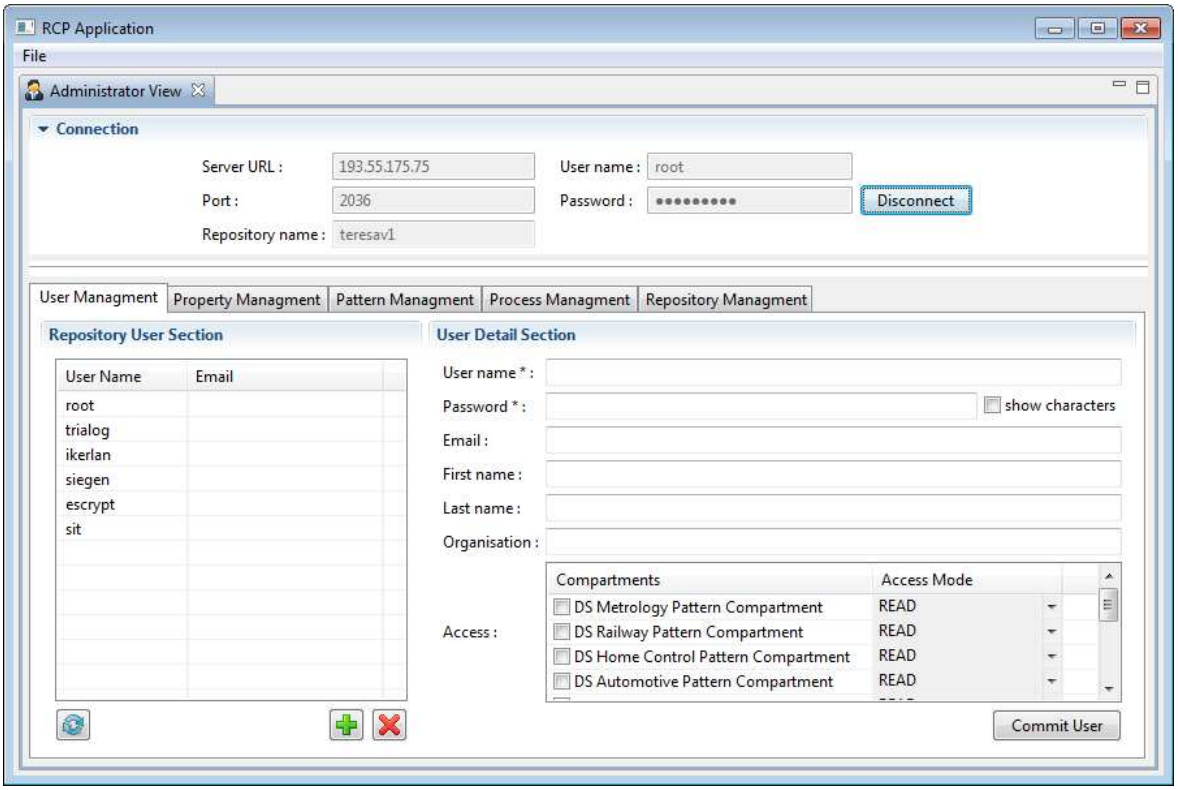

Figure 6.18: The Admin UI of the Repository

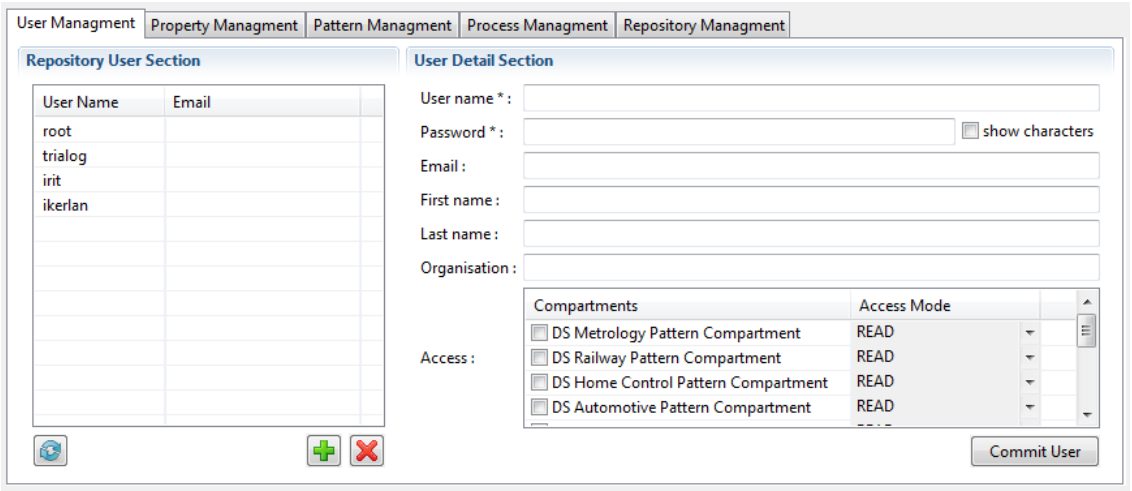

Figure 6.19: User Management Part

- In the host field, the address (URI) of the host should be entered. In our case **www.semcomdt.org** is used which is hosted by the University of Toulouse 2.

- In the Repository name field, the name of the repository should be entered

(for TERESA we used *teresav1*).

- In the User field, the username under which you want to connect to the repository must be entered.
- In the Password field, the password for the above username should be entered.

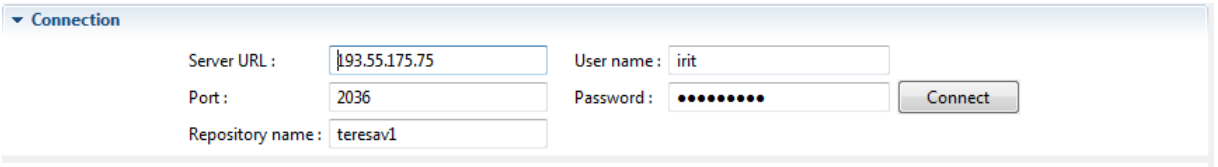

Figure 6.20: Repository Authentication

As the tool-suite is provided as Eclipse plugins, we also provide an authentication facility as an Eclipse preferences set as shown in Figure 6.21.

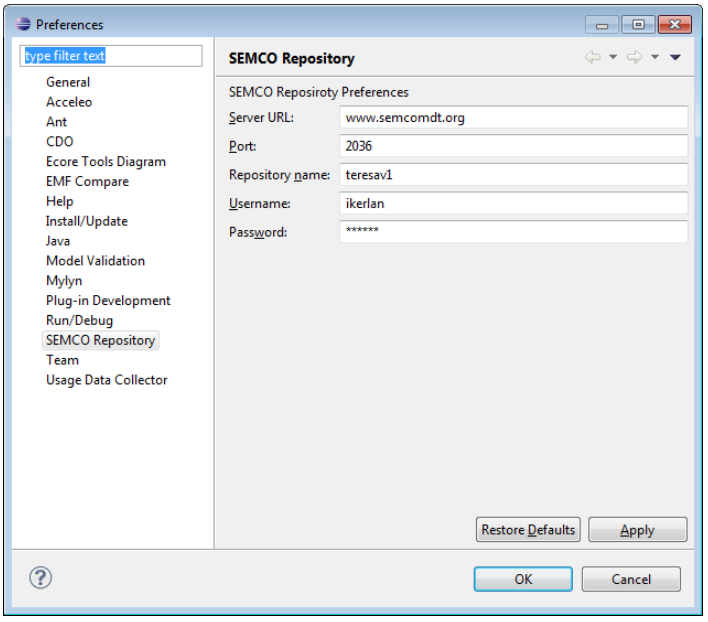

Figure 6.21: Repository Authentication Under Eclipse

6.6.2 Artifact Management

We provide a tool, as a Java based GUI application named *GayaAdmin* to manage relationships among S&D pattern specifications, to support the management of pattern system configuration, and between S&D patterns and their related property

models. For instance, as visualized in Figure 6.22, a pattern may be linked with other patterns and associated with property models using a predefined set of reference kinds such as those proposed in the SARM metamodel. Moreover, we support basic features such as artifact management. *GayaAdmin* uses the *Gaya4Admin* API.

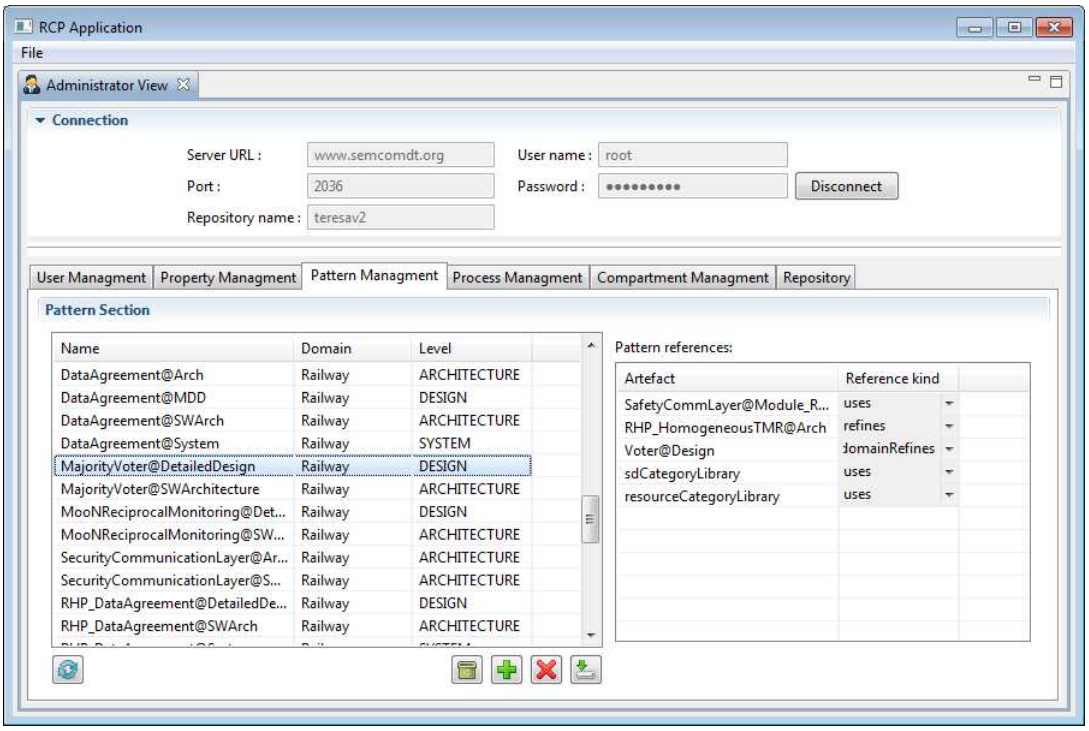

Figure 6.22: Repository organization

The *GayaAdmin* tool provides a set of dialogues to facilitate the management of the modeling artifacts available in the repository. The pattern management supports a set of features such as pattern lookup, removal, sorting, exporting and categorization as shown in Figure 6.24. The property management supports a set of features such as property library lookup, removal, sorting, exporting and categorization as shown in Figure 6.23.

6.7 Systems of Patterns Modeling

To model the system of patterns, *Arabion* tool provides a tree editor shown in Figure 6.25. In the left side, we distinguish three folders dedicated for: (1) Patterns modeling, (2) Systems of pattern modeling and (3) Configurations derived from systems of patterns. First, we use the access tool presented in 6.5 to search and import patterns in our workspace. Then, using *Arabion* tool, we create a new system

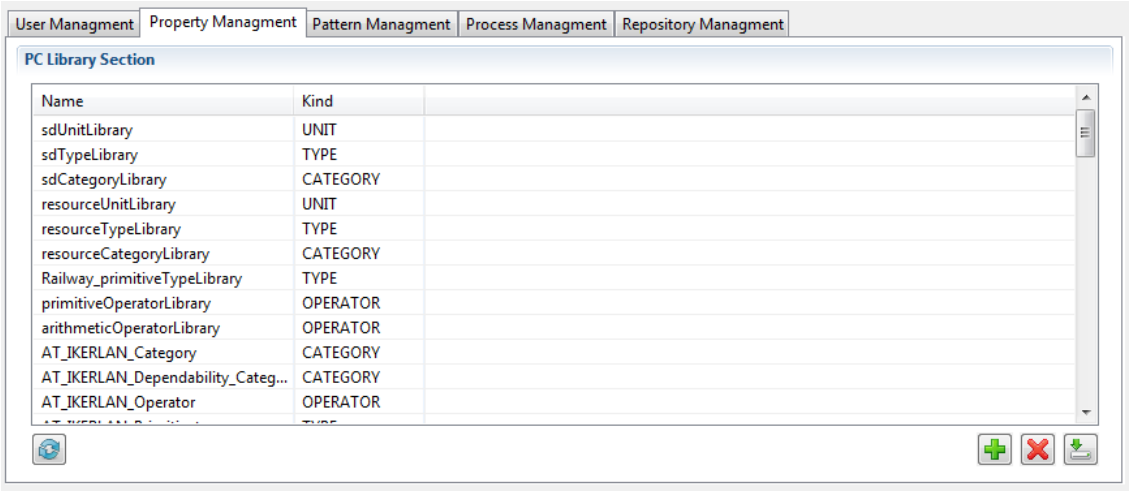

Figure 6.23: Property management part

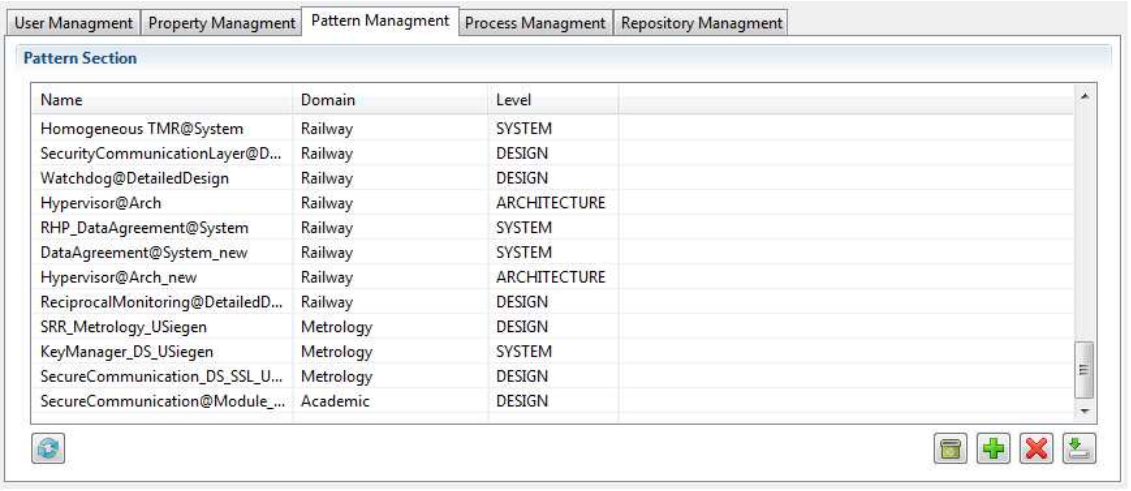

Figure 6.24: Pattern management part

of pattern model including the imported patterns those located in patterns folder (see Figure 6.26). Finally, we complete the model by adding the references between patterns; for each one we specified the name, kind, source pattern and target pattern as shown in the bottom of Figure 6.25.

The modeled system of patterns cannot be used, the way it is, as input for the analysis. This is because it includes some patterns that perform the same functionalities. For example, *SwMajorityVoter* and *HwMajorityVoter* are respectively the software variant and the hardware variant of the *MajorityVoter* pattern. The link between these two patterns is bidirectional and is of "isSimilar" kind. Based on this, we can imagine two different configurations with in each, a variant of the

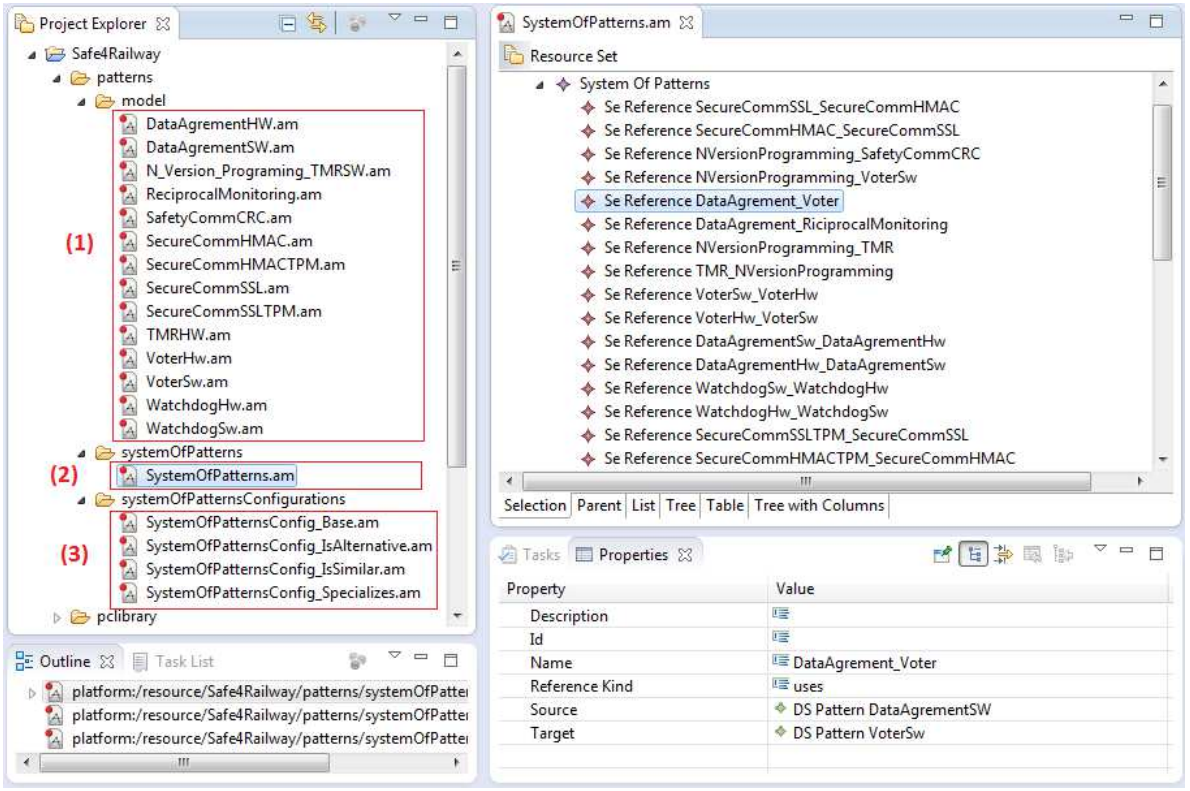

Figure 6.25: System of patterns modeling using Arabion tool

MajorityVoter is used. In the same way, the other kinds of links except "uses" allow to produce multiple pattern system configurations. Using the algorithm proposed in 4.4 with the system of all the patterns as input, we generate a set of configurations that will be simulated with the platform in order to select the ones that give the best resources consumption trade-off.

6.8 Conclusion

The proposed approach promotes a model-based developement coupled with a repository of models for embedded system applications, focusing on the problem of integrating non-functional properties by design to foster reuse. To fulfill part 3 of ***RG1*** and part 2 of ***RG2***, we have developed an MDE tool-chain based on Eclipse technology, mainly on Eclipse Modeling Framework Technologies (EMFT[6]) and a CDO-based repository. Currently, the tool suite named SEMCOMDT is provided as Eclipse plugins.

We build a new design environment supporting repository-centric PBSE ap-

[6]http://www.eclipse.org/modeling/emft/

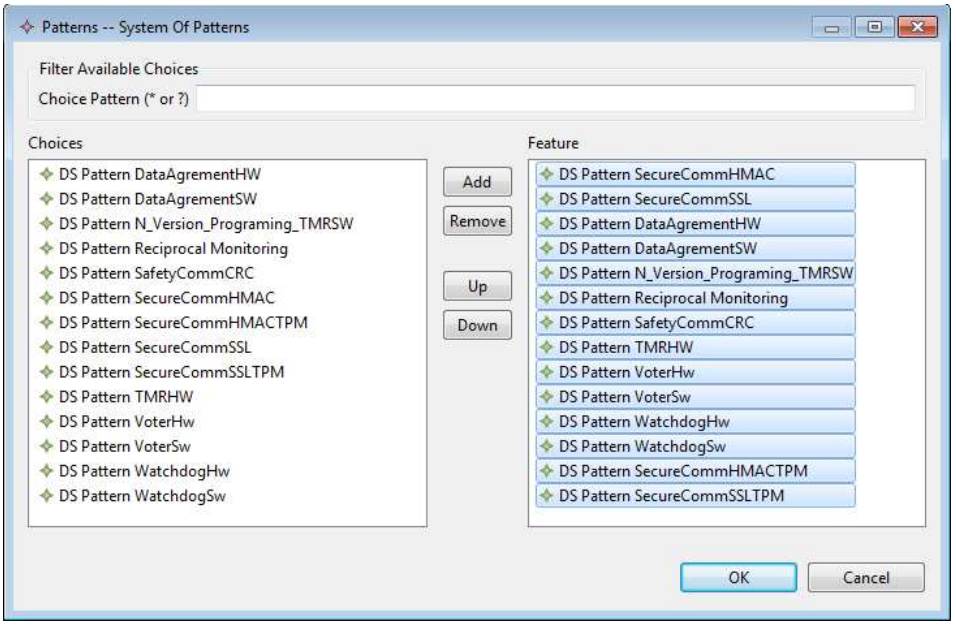

Figure 6.26: Definition of the system of patterns - Included patterns

proach for S&D applications development in RCES. The resulting repository system with its associated design environment, documentation and a number of guidelines, will facilitate 1) the population of the repository with further security and dependability patterns, and 2) the access to the repository and transformation of the S&D patterns into platform dependent specifications.

Chapter 7

Demonstration and Evaluation

7.1 Introduction

In this chapter, we demonstrate the applicability of our proposed framework (***RG3***) throw the *Safe4Rail* demonstrator developed in the context of TERESA project. The selection of an application to evaluate the proposed approach has to be considered as a very important decision for our work. It has a direct impact on the selection of patterns, and thus on the whole evaluation. The main characteristic that was considered to select the Safe4Rail application was the need to develop an S&D-related system that can meet a SIL[1] 4 level. This characteristic requires a number of design techniques as redundancy, diversity and monitoring to be taken into account while implementing the application.

The remainder of this chapter is organized as follows. In Section 7.2, we describe the demonstrator including the Safe4Rail platform and the Safe4Rail application requirements. Then, Section 7.3 gives an overview of the current TERESA repository content. In Section 7.4, we model the Safe4Rail application using the SEMCOMDT tool-chain while in Section 7.5, we provide a set of experimentations to analyze the resource consumption of the selected pattern system configurations. In Section 7.6, we evaluate the applicability of our pattern-based modeling approach. Finally, we discuss the power and the potential of our approach in Section 7.7.

[1]Safety Integrity Level

7.2 Description of the Demonstrator

The railway case study (called Safe4Rail) is a simplified version of a real ETCS (European Train Control System) (see Figure 7.1). ETCS is a signalling, control and train protection system designed to replace the many incompatible safety systems currently used by European railways, especially on high-speed lines. The main functionality of this demonstrator is to supervise that travelled speed and distance does not exceed authorised maximum values provided by the railway infrastructure. This limit is represented with a braking curve constructed from a movement authority and maximum speed profiles. In order to implement this functionality, the system is composed of multiple subsystems including the European Vital Computer (EVC) that executes the safety application, a set of odometry sensors and actuators. The odometry sensors provide the speed and acceleration of the train. With these values, the system must be able to calculate accurate speed and position values (odometry).

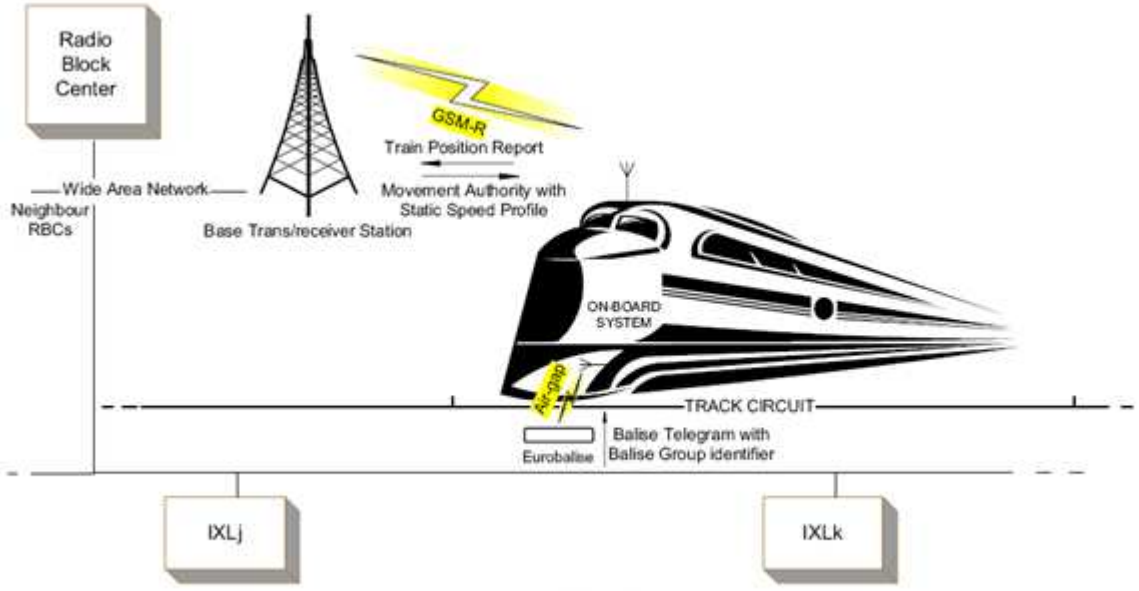

Figure 7.1: ERTMS/ETCS Level 2 diagram
[70]

The braking curve provides at every position three speed limits, which are used to make decisions about when to activate the brakes (see Figure 7.2):

1. When the current speed overcomes the warning speed limit, the system must activate a warning signal to advise the driver that the railway is getting close to a dangerous speed.

2. If the driver does not take any action and the railway overcomes the service speed limit, the system must activate the service brake.

3. If the train continues accelerating and overcomes the last limit, the system will deactivate the acceleration and activate the emergency brake to stop the train completely.

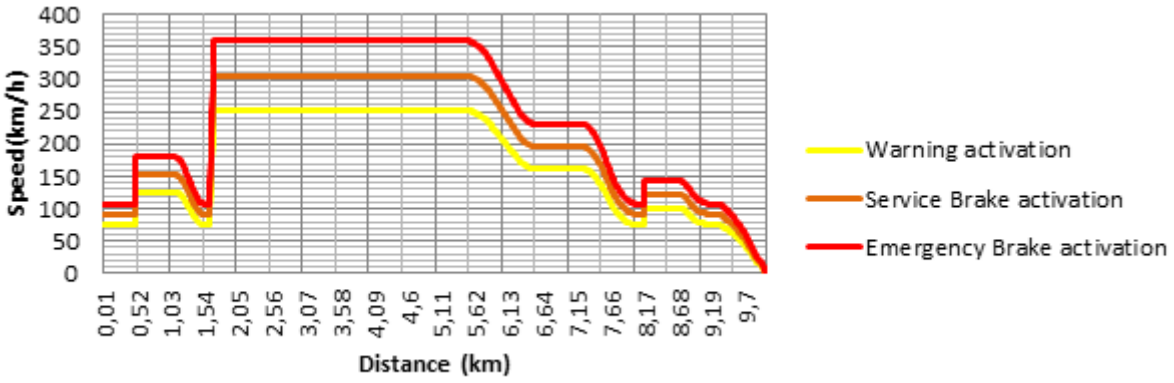

Figure 7.2: ERTMS/ETCS supervision limits and braking curves [70]

In the following sub-sections, we present the railway domain Safe4Rail platform and the application requirements, as described in the TERESA deliverables (D6.1 [73] and D6.3 [70]).

7.2.1 Description of the Platform

The following figures provide an overview on the platform's hardware components and their ports and interfaces. As shown in Figure 7.3 the hardware platform is composed of a carried board on which is installed a conga-CA board with a microprocessor (Intel Atom Z530), a RAM (DDR2 RAM) and a set of interfaces for the connections. The conga-CA is accompanied with an additional programmable resource calculation unit (FPGA-Spartran). For safety requirements, this platform is replicated thrice as shown in Figure 7.4. The three platforms are connected in star-topology to two Ethernet switches that execute the same application software. An additional platform executes a simplified model of the train (including sensors and actuators interface), majority voter and a simplified Driver Machine Interface.

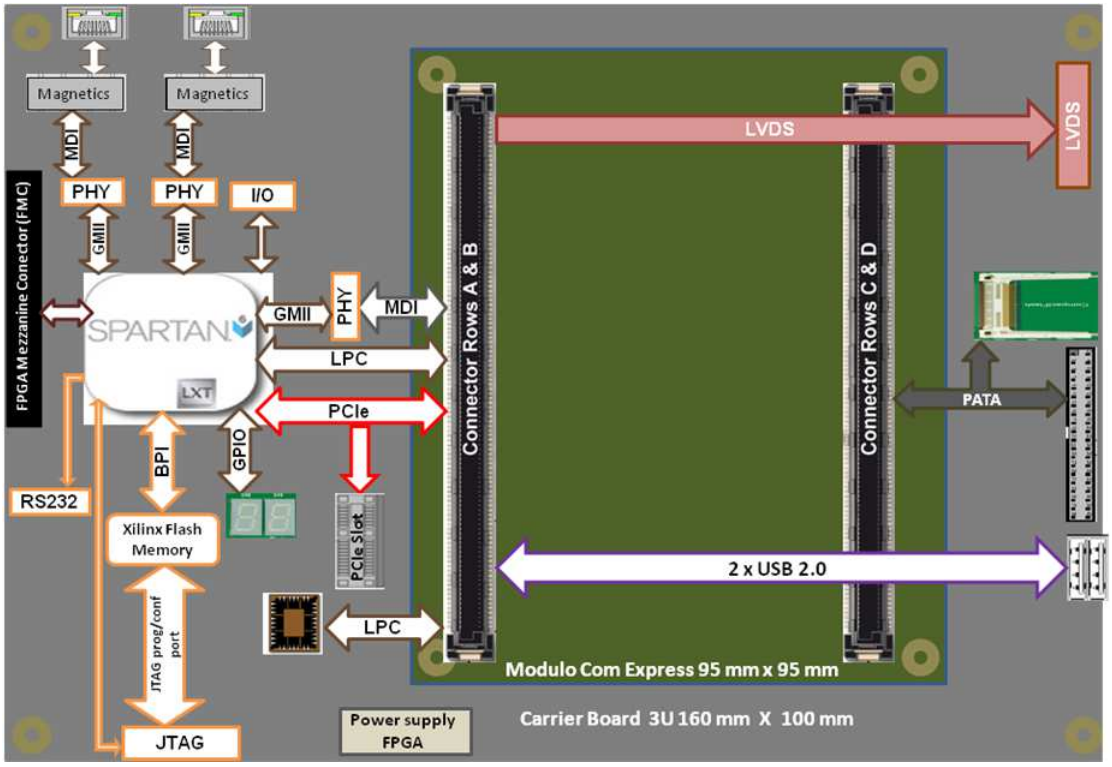

Figure 7.3: Hardware Platform Design

Figure 7.4: Architecture of the Safe4Rail Hardware Platform

7.2.2 Description of the Application

Figure 7.5 gives an overview on the whole system architecture before a description for each subsystem that contributes to the overlaying system is given. Furthermore the list provides the requirements which these components will fulfil.

- *Clock.* Generates a periodic event which triggers the system to estimate the current position and speed and to supervise that the train complies with the current track restrictions.

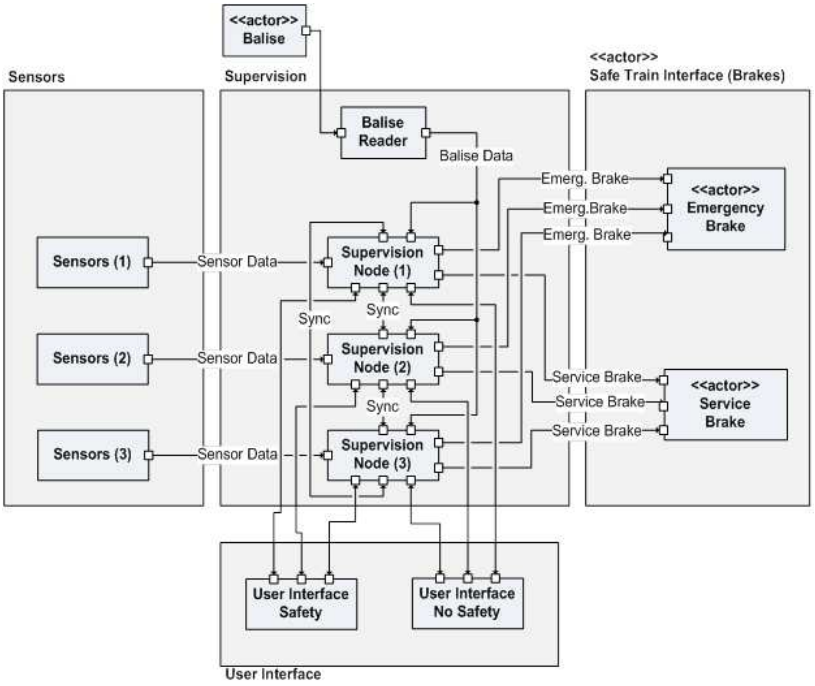

Figure 7.5: "Safe4Rail" System Components

- *Environmental Conditions.* Represent the physical interaction between environment (train, track, others) with the sensors of the system.

- *Balise.* Represents a Balise installed on the track which supplies to the train supervision system with new information regarding the current position and the track conditions.

- *Safe Train Interface.* Represents the actuators for the application.

- *Supervision System.*

  - *BaliseReader.* Detect and read the information provided by the balise on the rail.

  - *Supervision.* The main component of the system responsible of carrying out the functionality of the system.

  - *Sensors.* Provide the actual position and speed of the train and the track conditions to the system.

  - *User Interface.* The driver interacts with the system through this interface.

Non-Functional Requirements

In the Safe4Rail system there are three main use cases, which are described below. Figure 7.6 provides a diagram of selected use cases, showing their classification as safety-relevant and non-safety-relevant and their relationships.

1. Activate emergency brake and realize diagnostics (when the system is in Standby mode).
2. Supervise train speed and position (when the system is in Supervision mode).
   (a) Estimate current position and speed.
      i. Get sensor data.
      ii. Get balise data.
      iii. Calculate current position and speed.
   (b) Supervise the current position and speed and activate warnings and brakes accordingly.
      i. Process release emergency brake command.
   (c) Provide Information to the User.
3. Change between Standby and Supervision modes.

Based on the previous analysis, the following safety requirements are specified:

- (SIL4) Supervise train travelling speed and distance: The system shall supervise that train travelling distance and travelling speed does not exceed maximum safe authorized values, Movement Authority (MA) and speed profiles respectively.
  - (SIL4) Odometry: The system shall estimate traveling speed and distance with bounded absolute errors (ABS_DIST_ERR_MAX and ABS_SPEED_ERR_MAX for a maximum distance between eurobalises of DIST_MAX_BALISE and maximum speed of 500km/h).
  - (SIL4) Mode: The system shall safely manage modes and their transitions:
    * NO POWER: The system shall stay in safe state
    * STANDBY: The system shall stay in safe state

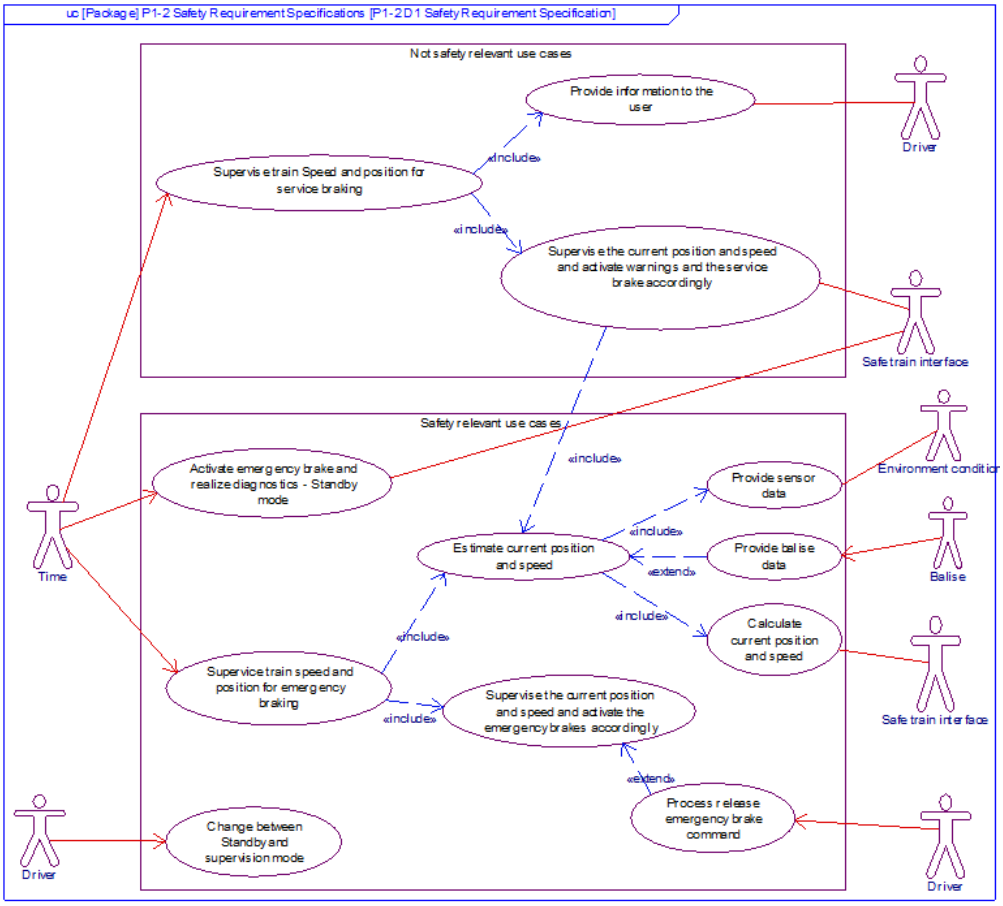

Figure 7.6: "Safe4Rail" Safety use-case diagram

* SUPERVISION: The system shall supervise that train travelling distance and travelling speed does not exceed maximum safe authorized values, Movement Authority (MA) and speed profiles respectively. (This implies the execution of multiple sub-safety functions such as 'communicate with control centers', 'limit supervision', etc.)

– (SIL4) Limit supervision: The system shall update maximum distance and maximum speed profiles with received commands, and compare estimated traveling distance and speed (odometry) with these limits. If any 'safe authorized limit' is exceeded (distance and/or speed) the safe state shall be activated.

– (SIL4) Rearm: Once the safe state is activated by 'limit supervision' safety function, the system can only be rearmed (release emergency brake) when the 'train is stopped' and the 'driver commands rearm'.

- The safe state is emergency brake activated (this will lead the overall system

to safe state, that is, train stopped).

- The fault-hypothesis is single fault support (non-byzantine).

The Architects analyze system safety requirements and mentally identify possible architectures and safety techniques to be used. They start defining the European Vital Computer (EVC). They identify *Triple Modular Redundancy (TMR)* as a design pattern of interest in order to reach a SIL4 by means of "composite fail-safety" technique. The TMR requires an external safety hardware *majority voter* implemented with two independent majority voters. Majority voters require six digital outputs of the TMR (three per majority voter and each digital output controlled by a different computation channel) to generate two independent emergency brake majority commands to the train-interface. Regarding the internal structure of the TMR, a black channel is selected to enable the communication among computation channels. Therefore a *Safety Communication Layer* pattern is already integrated. A *data agreement protocol* can be used in order to reach an agreement on the input values to be used by computation channels (input sensors are connected end-to-end to a single computation channel). This enables bit-exact execution of software that simplifies diagnosis. In order to avoid malicious intrusions a *Secure Communication Layer* pattern is integrated. Appendix B provides the full description of these patterns.

7.3 An Overview of the TERESA Repository Content

The TERESA repository contains so far (as of January 2013):

- *Property Libraries.* 69 property model libraries (see the left part of Figure 7.7):
  - 16 Unit Libraries
  - 23 Type Libraries
  - 20 Property Category Libraries
- *Pattern Libraries.* 59 pattern models (see the right part of Figure 7.7):
  - 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)

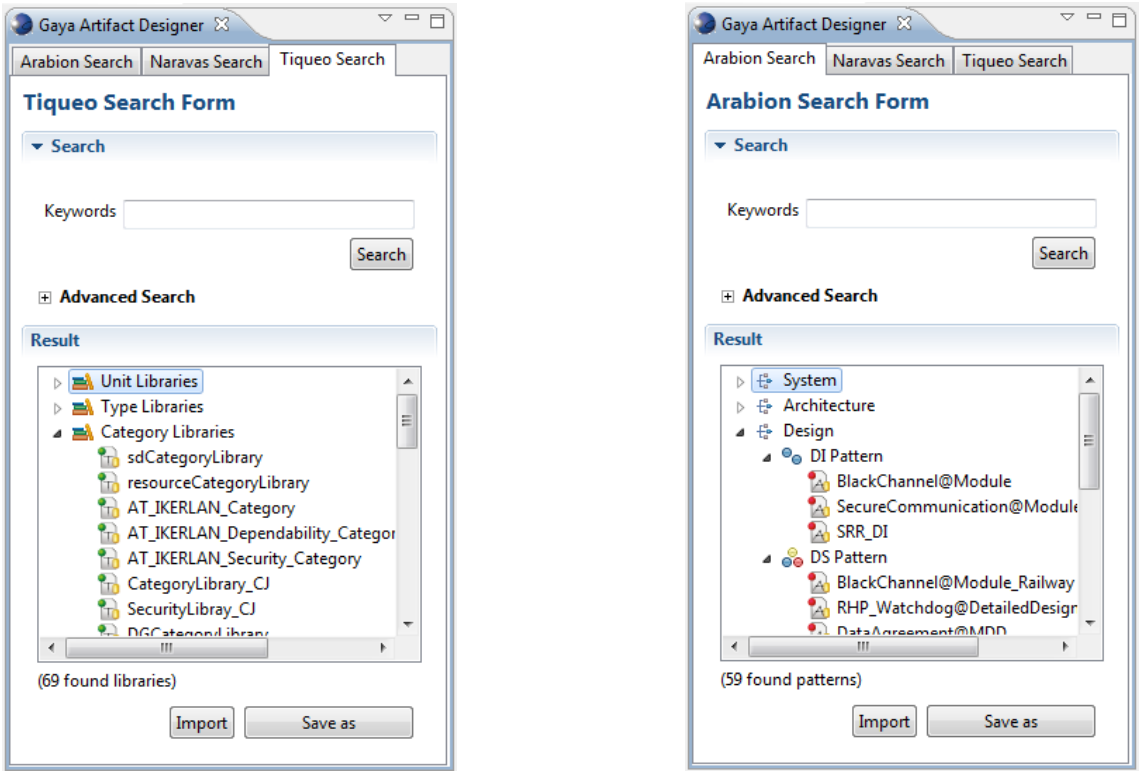

Figure 7.7: Repository Content

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.

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

Table 7.1: Railway Patterns

Pattern	*Security / Dependability*	*Domain Supported*	*Development Stage*
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

Table 7.2: Metrology Patterns

7.4 Modeling of Safe4Rail

This section deals with the modeling of the Safe4Rail platform and application using the tool-chain presented in chapter 6. First, we model the platform resources using the predefined libraries of resource categories. Then, the resulting platform model, is used as input to model the application based on the selected S&D patterns.

7.4.1 Safe4Rail Platform Modeling

In order to ease the definition of the platform model used along this study, we have used *Matho* which provides a tree editor for resources modeling as shown in Figure 7.8. The *Matho* allows to define the available platform by creating and incorporating the resources corresponding to each of the platform elements. These resources are tagged and configured using the *resource category libraries* and the *property category libraries* imported from the repository, so that the platform model can be built subsequently as a set of configured resources. The following table 7.3 gives a idea on how this platform is defined using the *Matho* tool.

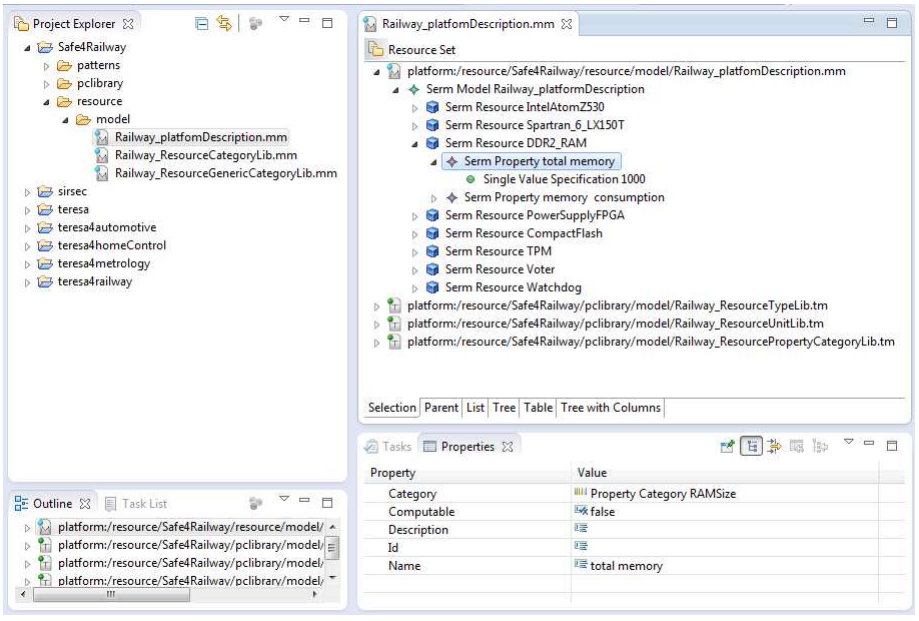

Figure 7.8: Safe4Rail platform description using Matho tool

Resource	Resource category	Property category	Resource property
IntelAtom Z530	CPU	CPUTime	Execution time reference
	CPU	CPUTime	Total CPU Time consumption
Spartran 6 LX150T	FPGA	FPGATime	FPGA execution time reference
	FPGA	FPGATime	FPGA time consumption
PowerSupply FPGA	PowerSupply	powerConsumption	Available powerSupply consumption
	PowerSupply	powerConsumption	powerConsumption
DDR2_RAM	RAM	ramsize	total memory
	RAM	ramsize	memory consumption
CompactFlash	AuxilaryMemory	memorysize	total of auxilary memory
	AuxilaryMemory	memorysize	auxilary memory consumption
TPM	TPM	TPMsize	Total of TPM memory
	TPM	TPMsize	TPM memory usage
	TPM	TPMtime	TPM execution time reference
	TPM	TPMtime	TPM time consumption
Voter	Voter	Votertime	Voter execution time reference
	Voter	Votertime	Voter time consumption
WatchdogHW	Watchdog	WatchdogTime	watchdog execution time reference
	Watchdog	WatchdogTime	watchdog execution time for timer management

Table 7.3: Hardawre architecture platform

7.4.2 Safe4Rail Application Modeling based on S&D Patterns

To fulfill the non-functional requirements identified previously in section 7.2.2 a set of patterns are selected and imported from the TERESA repository. Figure 7.9 show how this patterns are interconnect to form the whole pattern system. In the following, Table 7.4 and Table 7.5 give a description of the patterns. Note however, that we kept only the resource properties of the patterns for simplicity and because this part (i.e. resource properties) will be used for resource consumption analysis.

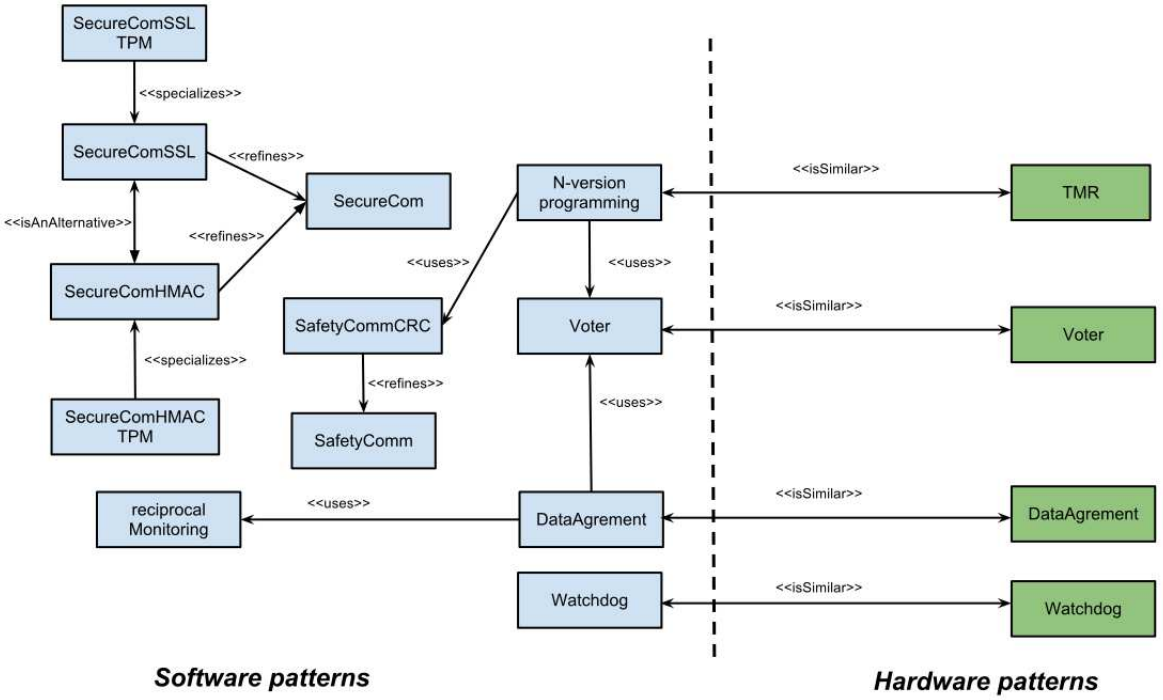

Figure 7.9: Architecture of complete pattern system

In order to model the application based on the selected patterns, we have used *Arabion* following modeling steps as explained in Section 6.7. This means that we have started by modeling the whole pattern system including all the patterns and the relationships between them as shown in Figure 7.9. Then, using the configuration generation algorithm introduced in Section 4.4, we get a set of pattern system configurations. Next, we will see how some of these configurations will be used for the resource consumption analysis.

Pattern	Resource category	Property category	Resource property
SecureCommSSL	CPU	CPUTime	CPU resource time for encryption
	RAM	ramsize	memory resource for encryption
	PowerSupply	powerConsumption	extra energy cost for encryption
	CPU	CPUTime	CPU resource for keyExchange
	RAM	ramsize	memory resource for keyExchange
	PowerSupply	powerConsumption	extra energy cost for keyExchange
	AuxilaryMemory	memorysize	memory resource for keyStorage
	CPU	CPUTime	CPU resource time for authentication
	RAM	ramsize	memory resource for authentication
	AuxilaryMemory	memorysize	memory resource for authentication keyStorage
	PowerSupply	powerConsumption	extra energy cost for authentication
SecureCommSSL TPM	CPU	CPUTime	CPU resource time for encryption
	RAM	ramsize	memory resource for encryption
	PowerSupply	powerConsumption	extra energy cost for encryption
	TPM	TPMsize	TPM capacity usage for keyMangament
	TPM	TPMtime	TPM execution time for keyMangament
	PowerSupply	powerConsumption	TPM extra enery cost for keyMangament
	TPM	TPMsize	TPM capacity usage for authentication
	TPM	TPMtime	TPM execution time for authentication
	PowerSupply	powerConsumption	TPM extra energy cost for authentication
SecureComm HMAC	CPU	CPUTime	CPU resource time for HMAC computation
	RAM	ramsize	memory resource for HMAC computaton
	AuxilaryMemory	memorysize	memory resource for HMAC storage
	PowerSupply	powerConsumption	extra enery cost for HMAC managment
SecureComm HMAC TPM	TPM	TPMsize	TPM capacity usage for HMAC managment
	TPM	TPMtime	TPM execution time for HMAC managment
	PowerSupply	powerConsumption	extra enery cost for HMAC managment
SafetyCommCRC	CPU	CPUTime	CPU resource time for CRC computation
	RAM	ramsize	memory resource for CRC computation
	PowerSupply	powerConsumption	extra enery cost for CRC computation

Table 7.4: Architecture of pattern system - Part 1

Pattern	Resource category	Property category	Resource property
VoterSW	CPU	CPUTime	CPU resource time for voting
	RAM	ramsize	memory resource for voting
	PowerSupply	powerConsumption	extra energy cost for voting
VoterHW	Voter	Votertime	voter execution time for voting
	PowerSupply	powerConsumption	extra energy cost for voting
WatchdogSW	CPU	CPUTime	CPU resource time for timer management
	RAM	ramsize	memory resource for timer management
	PowerSupply	powerConsumption	extra energy cost for timer management
WatchdogHW	Watchdog	WatchdogTime	watchdog capacity usage for timer management
	PowerSupply	powerConsumption	extra energy consumption for timer management
N-version programming (TMRSW)	CPU	CPUTime	CPU resource time for execution N software versions
	RAM	ramsize	memory resource for execution N software versions
	PowerSupply	powerConsumption	extra energy cost for execution N software versions
TMRHW	PowerSupply	powerConsumption	extra energy cost
Reciprocal Monitoring	CPU	CPUTime	CPU resource time for node checking
	RAM	ramsize	memory resource for node checking
	PowerSupply	powerConsumption	extra energy cost for node checking
DataAgrementSW	CPU	CPUTime	CPU resource time for acquiring inputs
	RAM	ramsize	memory resource for acquiring inputs
	PowerSupply	powerConsumption	extra energy cost for acquiring inputs
	CPU	CPUTime	CPU resource time for agreement processing
	RAM	ramsize	memory resource for agreement processing
	PowerSupply	powerConsumption	extra energy cost for agreement processing
DataAgrementHW	FPGA	FPGAtime	FPGA execution time for agreement processing
	PowerSupply	powerConsumption	extra energy cost for agreement processing

Table 7.5: Architecture of pattern system - Part 2

7.5 Analysis of Safe4Rail application

Among the generated configurations of our pattern system, we have selected only four to be analyzed. The idea is to give each time a different scenario with a configuration. Thus, the selected one are as following:

- Scenario 1: "Base" configuration (Figure 7.10).
- Scenario 2: "isSimilar" Configuration (Figure 7.11).
- Scenario 3: "isAlternative" Configuration (Figure 7.12).
- Scenario 4: "Specializes" Configuration (Figure 7.13).

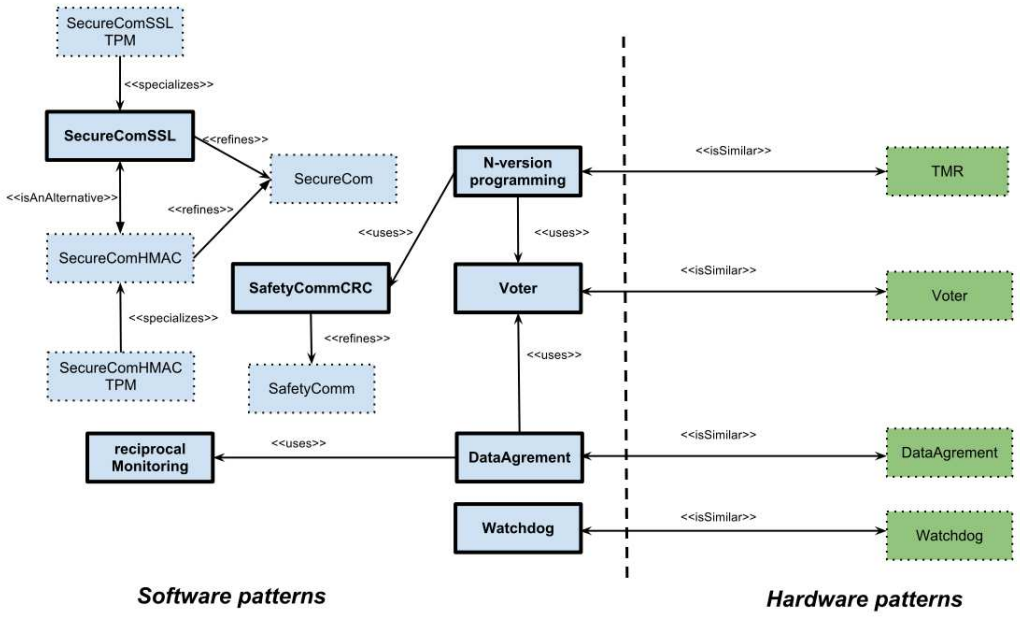

Figure 7.10: Pattern system base configuration

To describe the conduct of a scenario, we have selected the "isSimilar" configuration. As shown in Figure 7.11, in this configuration we have replaced the software voter by a hardware voter and calculate the impact on the resource consumption of this system of patterns. Both the system of patterns and the platform are modeled as shown previously. To calculate the whole resource consumption of the pattern system, first, we perform the model-to-model transformation. This takes as an input the pattern system model and the platform model and produces new platform

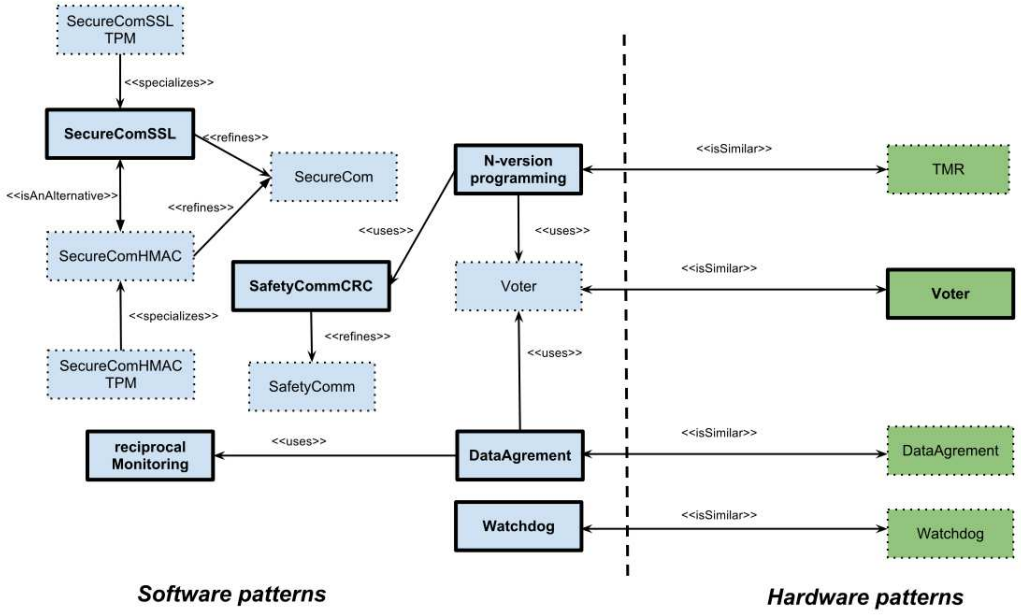

Figure 7.11: IsSimilar pattern system configuration

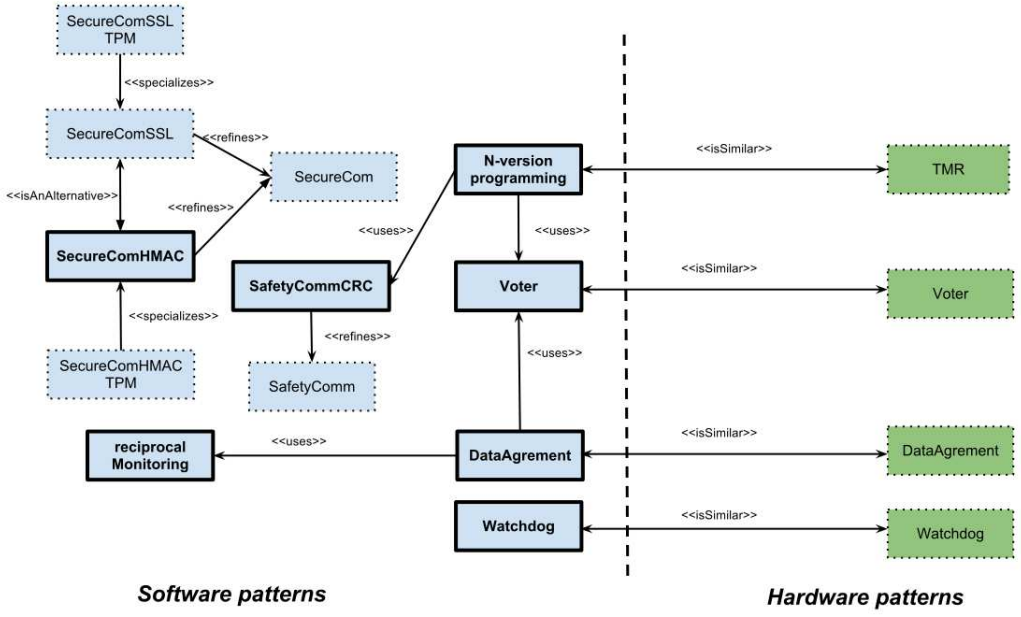

Figure 7.12: IsAlternative pattern system configuration

model annotated with the calculated values of the resource consumption (see Figure 7.14). Then, we perform the model-to-text transformation to produce HTML file for visualizing the result of the previous calculation (see Figure 7.15).

Table 7.6 and Figure 7.16, give a comparative view of the resource consumption of the four pattern system configurations.

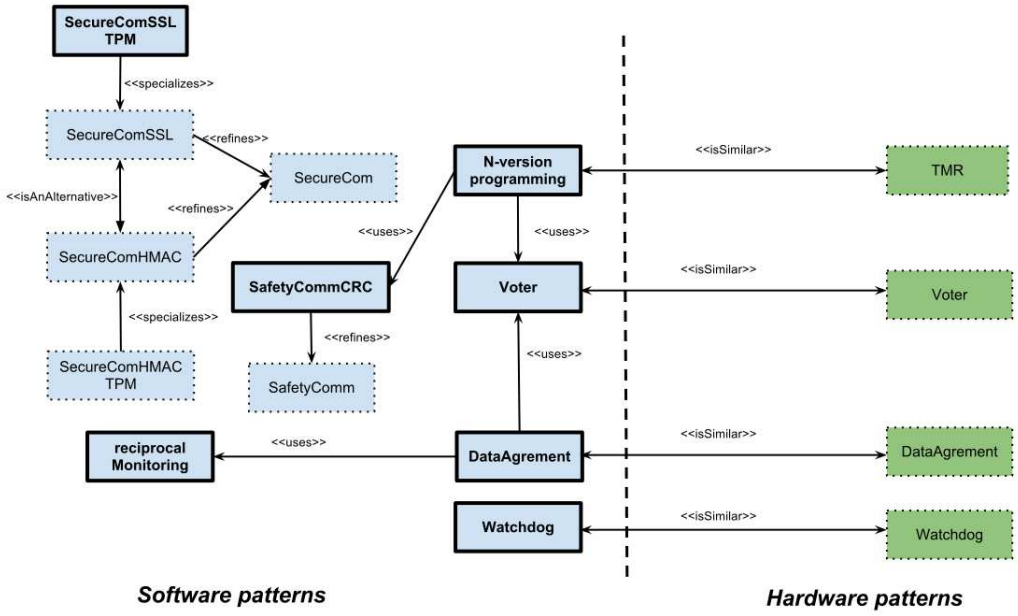

Figure 7.13: Specializes pattern system configuration

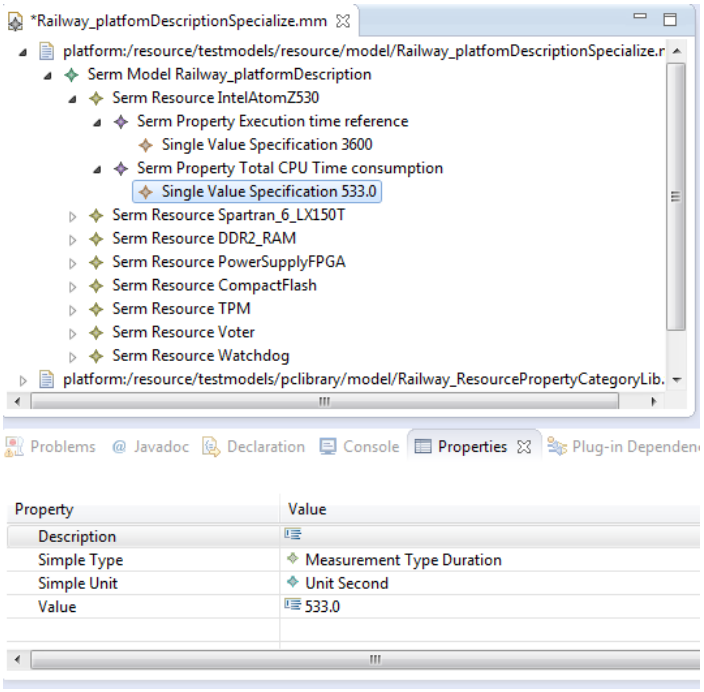

Figure 7.14: Calculation of the resource consumption - Scenario 2 (M2M transformation)

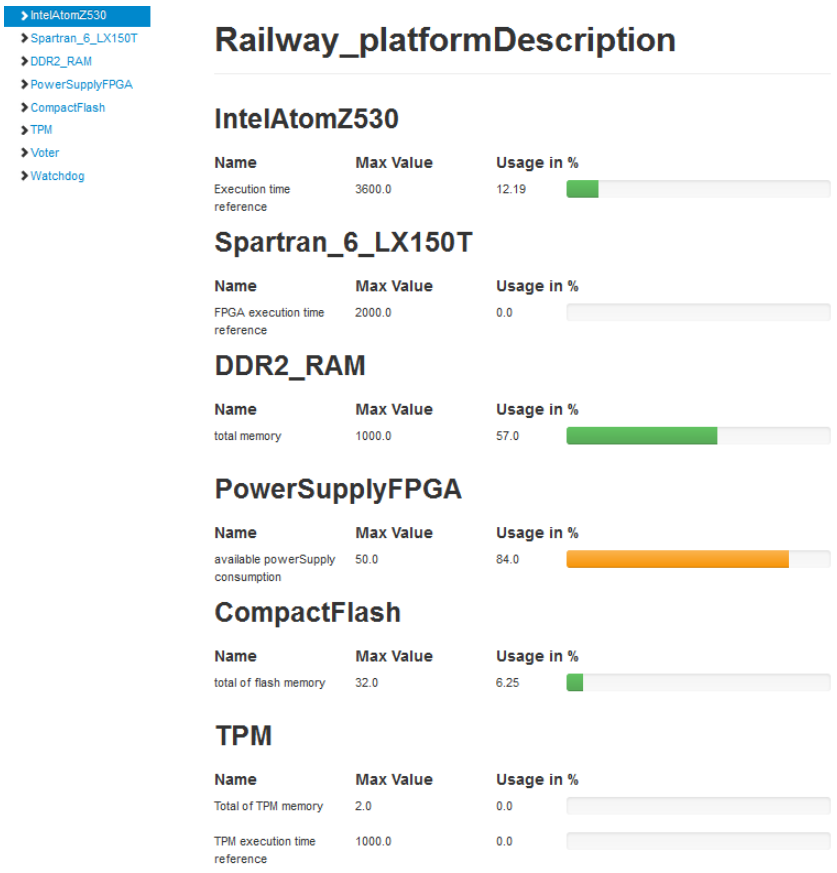

Figure 7.15: Visualization of the resource consumption - Scenario 2 (M2T transformation)

			Scenario 1		Scenario 2		Scenario 3		Scenario 4	
Resource	Property category	Max value	Usage	Usage in %	Usage	Usage in %	Usage	Usage in %	Usage	Usage in %
IntelAtomZ530	CPU Time (s)	3600	439	12,19%	379	10,53%	339	9,42%	533	14,81%
Spartran_6_LX150T	FPGA Time (s)	2000	0	0,00%	0	0,00%	0	0,00%	0	0,00%
DDR2_RAM	RAM Size (MB)	1000	570	57,00%	522	52,20%	398	39,80%	398	39,80%
PowerSupplyFPGA	Power consumption	50	42	84,00%	44	88,00%	36	72,00%	50	100,00%
CompactFlash	Memory Size (MB)	32	2	6,25%	2	6,25%	4	12,50%	0	0,00%
TPM	TPM Size	2	0	0,00%	0	0,00%	0	0,00%	1	50,00%
	TPM Time	1000	0	0,00%	0	0,00%	0	0,00%	50	5,00%
Voter	Voter Time	500	0	0,00%	50	10,00%	0	0,00%	0	0,00%
Watchdog	Watchdog Time	200	0	0,00%	0	0,00%	0	0,00%	0	0,00%

Table 7.6: Analysis of the four scenarios

7.6 Evaluation

7.6.1 Context and Description of the Methodology for Experimentation

This section provides a preliminary evaluation of the approach along ISO-9126's quality-in-use dimensions, i.e. effectiveness, productivity, safety and satisfaction.

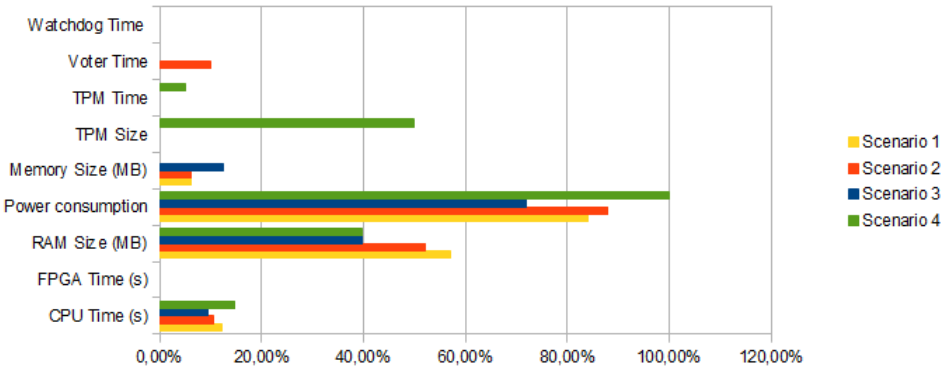

Figure 7.16: Analysis of the four scenarios - Graphic

Eleven TERESA members participated. They were handed out a sheet with instructions for each task (e.g. what properties to specify and what patterns to develop, when to take note of the time).

The study was divided into three tasks. Before they started, a general description of the aim of the study was given. Some running examples were introduced to them. After these two tasks, achieved during the TERESA MDE workshop in Toulouse (April 2012), a 6-months evaluation was conducted.

All the subjects were already familiarized with MDE, S&D patterns and Eclipse, though some did not know some of the companion plugins (e.g. Acceleo). Hence, the generation of documentation was not part of the evaluation. The procedure includes five tasks: SEMCO plug-in installation, property models development, pattern development, patterns instantiation and patterns integration.

7.6.2 Results

Effectiveness

Figure 7.17 shows a table providing the fulfilment for the five tasks. One subject had problems in using UML editors (Rhapsody or Papyrus) for pattern integration, and hence, he was excluded from the rest of the experiment.

Productivity

Productivity is measured as the number of minutes required for each task (only for those that successfully completed the first task): SEMCO plugin installation took between 10 and 15 minutes, with a mean of 12; property model development took between 20 and 60 minutes, with a mean of 42.5 minutes; pattern development

Item	Frequency	%
Task 1. Plugin installation	3.5	100
Task 2. Property Model development	5.5	100
Task 3. Pattern development	10	100
Task 4. Pattern instantiation	11.2	100
Task 5. Integration with other tools	11.2	90

Figure 7.17: Effectiveness Results

took between 40 and 60 minutes, with a mean of 53 minutes; pattern instantiation took between 10 and 30 minutes, with a mean of 19 minutes and finally, pattern integration took between 30 and 120 minutes, with a mean of 61 minutes.

Item	Mean (minutes)	St. Dev.
Task 1. Plugin installation	12	2.8
Task 2. Property Model development	42.5	11.5
Task 3. Pattern development	53	8.4
Task 4. Pattern instantiation	19	8.8
Task 5. Integration with other tools	61	35.4

Figure 7.18: Productivity Results

User Satisfaction

Satisfaction is the capability of the software product to satisfy its users. In this case, the product is the repository of S&D patterns engine, and its ability to develop a trusted RCES application. We asked participants to give scores from 1 to 5 (5 is the best) and comments. We first evaluated the perceived usefulness of the solution itself (items 1-5). Next, we focus on the tool-suite as a mean to build the modeling artifacts. We separately collected the satisfaction along the four tasks (items 6-14). Finally, we want also to measure the willingness to use repository of modeling of S&D patterns in the future in the related activities (items 15-20). The following table depicts an overview of the results of our experiment.

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.

Analysis of Results

A 6-month evaluation identified the following advantageous features:

- For Design:

Item	Mean	St. Dev.
1. I think 'model based repository' is a good idea	4.90	0.18
2. I think 'model based repository' helps to keep focus without being distracted by other aspect of software engineering	4.40	0.48
3. I think 'model based repository' are useful for defining meaningful 'units of solution'	4.40	0.48
4. I think 'model based repository' save me time to develop S&D Embedded Systems	4.50	0.48
5. I think 'model based repository' avoids re-inventing existing solutions	4.40	0.50
6. I think the installation of the SEMCO plug-in is easy	4.60	0.48
7. I think repository populating tools are easy to use	4.10	0.48
8. I think repository access tools are easy to use	3.80	0.54
9. I think it is easy for me to develop new S&D patterns	4.10	0.68
10. I think it is easy for me to develop new property models	3.50	0.36
11. I think S&D patterns instantiation is easy to use	3.80	0.70
12. I think properties models instantiation is easy to use	3.80	0.64
13. I think S&D patterns integration is easy to use	3.50	0.64
14. I think property models integration is easy to use	3.40	0.60
15. I would like to develop S&D patterns in the future	4.60	0.60
16. I would like to develop properties models in the future	3.80	0.56
17. I would like to install other SEMCO plugins in the future	4.10	0.68
18. I would like to exchange SEMCO in the future	3.50	0.54
19. I would like to customize some SEMCO plugin in the future	3.60	0.56
20. I would like to extend some SEMCO features in the future	3.60	0.76
	3.70	0.83

Figure 7.19: Satisfaction Results from 1 (total disagreement) to 5 (total agreement).

  - Pattern language support
  - Helpers for pattern selection
  - Combined textual/graphical input
  - Data/Property typing : libraries
  - Multiple design environment support

- For V&V:
  - Validation at design time
  - Model validation
  - Guidelines for pattern and model integration

These results suggest that subjects like the notion of model-based repository as a way to speed the development of S&D applications by design (e.g. reuse existing

solution through pattern), and in so doing, improving focus on tough tasks (e.g. implementation). However, pattern integration stands up as the main stumbling block for pattern-based system development adoption. More to the point, if we consider that the subjects were programmer natives (i.e. accustom to use programming language for security engineering). Specifically, users tend to overlook the four rules that govern pattern-based system development (i.e. (1) each pattern must be specified domain-application independently, (2) more than one pattern is required to fulfill one S&D property, (3) every pattern should be instantiated in the target domain-development environment, and (4) every pattern should be integrated in the context of the under development system).

Our future work includes envisaging mechanisms that assist users in obtaining scenarios along these four rules. This is far from trivial as we have to deal with pattern integration, which are often a domain-development environment specific.

7.7 Synthesis and Discussion

The pattern, properties and resources modeling languages used in this thesis have evolved compared to ones presented in [31] and [87], respectively. 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 out the major industrial requirements that the framework now meets. One of them is the repository storage and interactions support 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 leads 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.

7.7.1 Recapitulation and Perspectives

The approach empowers the embedded systems engineer to reuse solutions for resource, security and dependability, without specific knowledge on how the solution is designed and implemented. This enables to work at a higher level of abstraction, which may significantly reduce the cost of engineering the system.

The resulting repository prototype, 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. Further on, the framework will be completed targeting a set of additional concerns, including:

1. S&D pattern modeling to get a common representation of patterns for several domains in the context of RCES (1) to capture the essence of the pattern, (2) to provide enough detail to improve the usability of the pattern by a non-specialist, (3) to provide sufficient information to be validated, (4) to provide sufficient explanation to improve the usability of the pattern in other domains as well as the domain in which the pattern was defined.

2. Patterns validation to enhance the pattern quality using formal validation approaches before publishing them in the repository. The resulting validation artifacts will be used during the pattern integration, guaranteeing the correctness of this step, as well as for the implementations with automatic derived guidelines for platform dependent implementations.

The expected goal is to highlight the content of the repository in the form of a map representing pattern dependencies. The proposed pattern and properties specification languages supports the specification of S&D patterns and their related properties, mainly S&D and resource properties. In addition to that, the languages may be used to specify other kind of patterns. For instance, memory, concurrency and distributed patterns [58].

In a wider scope, new specification languages may be designed and stored with their related artifacts in the repository. For instance, components, resources, analysis and simulation are important kinds of artifacts that we can consider in our framework to serve systematic construction of large complex systems with multiple concerns. 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, the tool suite promotes the separation of concerns during the development process by distinguishing the roles of the stakeholders. Mainly, the accessing to the repository is customized regarding the development phases and the stakeholders domain and system knowledge.

7.7.2 Limits of the Approach

In the presented approach, different levels of abstraction are used to get a common representation of patterns for several domains. In RCES concerns, systems include a combination of hardware and software components. This add some difficulties to build a simple modeling framework. A high level of abstraction is proposed to represent S&D patterns to capture aspects of security and dependability in the different domains of RCES, not an implementation of a specific solution. Other issues are:

- The repository is generated from existing metamodels. The question raised is whether the proposed structure can support evolution and dynamic specification languages.
- Access control policy is delegated to the underlying platform implementation. In our case, this policy is organized around compartment not around the artifact.
- The instantiation uses model transformation techniques. We have investigated a UML based IDE. The question raised is whether our proposed framework can support new development environments not yet covered by the repository Access Tool and the existing model transformations.
- In order to integrate a pattern in a system (application), some significant additional information about the pattern is required. For instance, the interfaces and their requirements. The goal is to capture how the system interacts with the patterns, and how the internal structure of the pattern interacts with other patterns in the case of composite patterns. Especially, when dealing with software and hardware components.

Chapter 8

Conclusion

Security and dependability requirements are incorporated to an increasing number of systems. These newer restrictions make the development of those systems more complicated than conventional systems [59]. The research community is seeking approaches to ease the design of these systems through reuse techniques that empower the designer to generate a system or part of it from a set of common and reusable artifacts.

The integration of S&D features requires the availability of both application domain specific knowledge and S&D expertise at the same time. An integrated tool-supported approach for capturing and providing this expertise by the means of S&D patterns is proposed in this thesis. Special emphasis will be devoted to promote the particularly challenging task of efficiently integrating security and dependability solutions within the restricted available design space for embedded system. Furthermore, one important focus is on the potential benefits of the combination of Model-Driven Engineering with a pattern-based representation of security and dependability solutions.

8.1 Summary and Contributions

Model-Driven Engineering (MDE) provides a very useful contribution to the design of RCES applications since it bridges the gap between design issues and implementation concerns. MDE can potentially maintain the separation of concerns between application and S&D, by ensuring that S&D designs can be reused at a later stage by application designers. Significant research is being carried out concerning MDE for embedded systems, at the level of system architecture, design techniques,

testing, validation, proof of correctness, modeling, software reliability, operating systems, parallel and real-time processing. But still research is needed on the use of MDE to enforce the integration of S&D requirements into the engineering process and to support the reuse of S&D mechanisms. One important focus in this field is on the potential benefits of the combination of Model-Driven Engineering with pattern-based representation of security and dependability solutions. Particular interest is given to the development of models and tools to support the inclusion of S&D issues into the RCES engineering process.

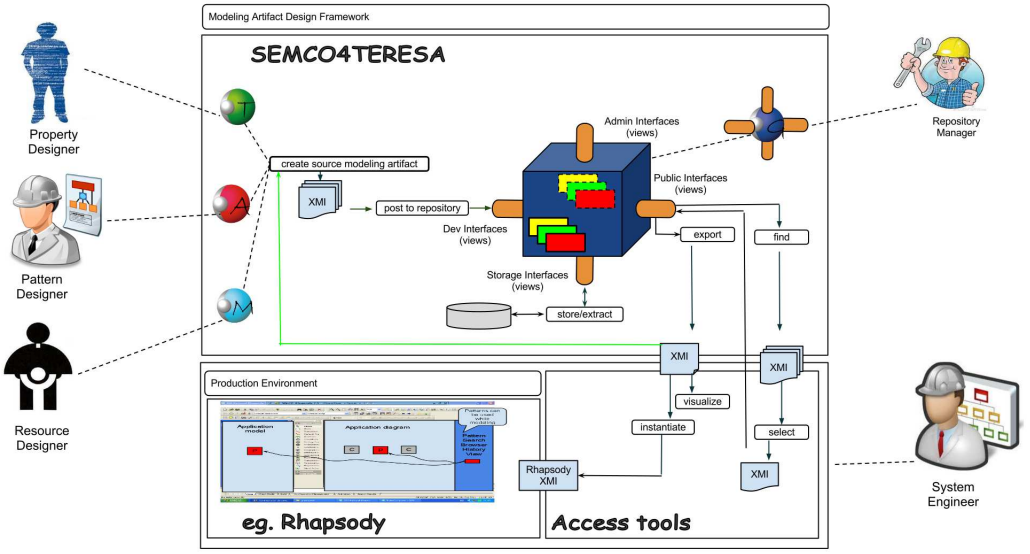

Figure 8.1: Tool-flow of the MDE-tool suite

The proposed approach promotes a model-based development coupled with a model-based repository (part 1 of ***RG2***) for RCES applications, focusing on the problem of integrating non-functional properties by design to foster reuse. The main goal of the repository is to share expertise, interacting with existing engineering processes in order to build trusted applications for Resource-Constrained Embedded Systems. The development of a model-based repository of modeling artifacts that follows the MDE paradigm is targeted in this thesis. Our framework is based on metamodeling techniques that allow to specify these modeling artifacts (part 1 of ***RG1***) at different levels of abstraction and an operational architecture of the repository.

Furthermore, we walk through a prototype with EMF editors and a CDO-based repository supporting the approach. Currently the tool suite named SEMCOMDT is provided as Eclipse plugins. An example of scenario of using the proposed integrated

set of tools is visualized in Figure 8.1. The developed tool suite aims to support the development, the management and the use of the modeling artifacts discussed in this work through (part 3 of ***RG1*** and part 2 of ***RG2***):

1) a framework to specify the modeling artifacts using Eclipse technologies (EMF, GMF, CDO, Papyrus) to populate the model-based repository (part 3 of ***RG1***), and

2) a repository of integrated models (patterns, S&D models, ..) and a repository access tool to allow application designers to capitalize on the MDE even if they are not experts in modeling (part 2 of ***RG2***), and

3) a set of transformation engines to adapt the representation of S&D patterns into platform dependent specifications. Moreover, we provide a set of transformation that can be useful in the analysis of software architecture based on patterns ((part 3 of ***RG1***).

The approach presented here has been evaluated with the Safe4Rail application in the context of the TERESA project for RCES application based on S&D patterns (***RG3***). For instance, a pattern designer defines patterns and stores them in the repository. A system designer reuses existing patterns from the repository through instantiation mechanisms which leads to simpler and seamless designs with higher quality and costs savings. By this illustration, we can validate the feasibility and effectiveness of the proposed specification and design frameworks.

8.2 Limitations and Future work

As a side remark, note that the goal is to obtain an even higher level abstraction to represent S&D patterns to capture several facets of security and dependability in the domain of embedded system applications, not an implementation of a specific solution. The key is then to show that the major sectors of trusted embedded system applications dealing with security and dependability are covered by our approach. This result leads to some anticipated issues about general techniques to model S&D patterns. It is of particular interest to build a trusted computing engineering discipline that is suited to a number of sectors in embedded systems. An important point concerns the completeness of the DSLs to support the evolution of the already stored artifacts and to support the representation of new modeling artifacts. Thus, if a domain specific artifact is missing, the repository supports extension mechanisms which allows the user to add the needed artifact.

First evidences indicate that users are satisfied with the notion of 'model-based

repository of S&D patterns'. The approach paves the way to let users define their own road-maps upon the PBSE methodology. First evaluations are encouraging with 85% of the subjects being able to complete the tasks. However, they also point out one of the main challenges: automatic search for the user to derive those 'S&D patterns' from the requirements analysis. We plan to perform additional case studies to evaluate both the expressiveness and usability of the methodology, the DSLs and the tools. Our vision is for 'S&D patterns' to be inferred from the browsing history of users built from a set of already developed applications. The next steps of this work consist of:

- Pattern System. The expected goal is to highlight the content of the repository in the form of a map representing pattern dependencies. In contrast, the proposed pattern and properties specification languages supports the specification of S&D patterns and their related properties, mainly S&D and resource properties. We are studying more sophisticated techniques for pattern system building. In addition to that, the languages may be used to specify other kind of patterns. For instance, memory, concurrency and distributed patterns [58].

- Modeling artifacts. New specification languages may be designed and stored with their related artifacts in the repository. For instance, components are important kind 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 artifacts 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, accessing the repository is customized regarding the development phases and the stakeholder's domain and system knowledge.

- Repository. All patterns are stored in a repository and organized to support horizontal and vertical relationships between patterns. Thanks to this, it is possible to find a pattern with an S&D criteria and previous design decisions. Moreover, we plan to study the automation of the search and instantiation of models and patterns and a framework for simpler specification of constraints would be beneficial.

- PBSE. We will seek integrating all the presented results in a more global process within the pattern lifecycle (i.e. create, update, store patterns) and the integration of a pattern in the whole application development lifecycle stages. Finally, guidelines will be provided and stored in the repository (during the pattern development) and reused (during the application development) (i.e. help to select a suitable pattern with respect to the constraint and the specificity of the target application and/or platform).

- Tool suite. With regard to the tool suite, we plan the development of a graphical and/or textual DSL editor on the one hand and the use of the QVT transformation rules as part of its functionalities. In addition, we will study the integration of our tooling with other MDE tools, mainly those used in RCES system development. For that, code generators need to implemented to generate a restrictive set of code complying to the domains standards. With regard to the tool-suite dissemination, currently the repository and the design environment are provided as an Eclipse plugin and for the near future we investigate an additional implementation exposing web services for development environments. We will also seek new opportunities to apply the framework to other domains.

- Validation. We plan to perform additional case studies to evaluate both the expressiveness and usability of the methodology, the DSLs and the tools. We will seek new techniques for the automation of the proposed methodology and the related tool suite.

Bibliographie

[1] . CDO Model Repository Overview. http://www.eclipse.org/cdo/. 19, 82

[2] . Qvt operational language. https://projects.eclipse.org/projects/modeling.mmt.qvt-oml. 19

[3] J-M. Jézéquel A. L. Guennec, G.n Sunyé. Precise modeling of design patterns. In *In Proceedings of UML?00*. Springer-Verlag, 2000. 32

[4] M. Hansen A. Pfitzmann. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. 33, 2010. 13

[5] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*, volume 2 of *Center for Environmental Structure Series*. Oxford University Press, New York, NY, 1977. 15, 32

[6] Hany H. Ammar, Vittorio Cortellessa, and Alaa Ibrahim. Modeling resources in a uml-based simulative environment, 2001. 31

[7] A. Avizienis, J-C Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1:11–33, 2004. 12

[8] Lothar Baum and Thorsten Kramp. Towards a uniform modeling technique for resource-usage scenarios. In *PDPTA*, pages 1324–1329, 1999. 31

[9] Philip A. Bernstein and Umeshwar Dayal. An Overview of Repository Technology. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 705–713. Morgan Kaufmann Publishers Inc., 1994. 21, 34, 61

[10] J. Bézivin. Towards a precise definition of the omg/mda framework. In *Proceedings of ASE'01*, pages 273–280. IEEE Computer Society Press, 2001. 1, 14

[11] R Birukou, E. Blanzieri, P. Giorgini, and M. Weiss. Facilitating pattern repository access with the implicit culture framework. In *Proceedings of "EuroPLoP 2007"*, 2007. 35

[12] F. Buschmann, K. Henney, and D. Schmidt. *Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing.* Wiley, 2007. 11

[13] G. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: a system of patterns*, volume 1. John Wiley and Sons, 1996. 16

[14] I. Crnkovic. Component-based software engineering—new challenges in software development. *Software Focus*, December 2001. 1, 40

[15] I. Crnkovic, Michel R. V. Chaudron, and S. Larsson. Component-based development process and component lifecycle. In *Proceedings of the International Conference on Software Engineering Advances (ICSEA 2006)*, page 44. IEEE Computer Society, 2006. 1, 40

[16] S. Ghosh D-K Kim, R. France and E. Song. A UML-based meta-modeling language to specify design patterns. In *Patterns, Proc. Workshop Software Model Eng. (WiSME) with Unified Modeling Languages*, 2004. 32

[17] J. Grundy D. Mapelsden, J. Hosking. Design pattern modelling and instantiation using DPML. In *CRPIT '02: Proceedings of the Fortieth International Conference on Tools Pacific*, pages 3–11. Australian Computer Society, Inc., 2002. 33

[18] F. Daniels. The reliable hybrid pattern: A generalized software fault tolerant design pattern. In *Proc. of the Pattern Language of Programs (PLoP'97)*, 1997. 17, 32, 33

[19] A. H. Eden E. Gasparis, J. Nicholson. LePUS3: An Object-Oriented Design Description Language. In *In: Gem Stapleton et al. (eds.) DIAGRAMS, LNAI 5223*, page 364?367, 2008. 33

[20] Eclipse RCP. http://www.vogella.de/articles/EclipseRCP/. 19

[21] R. B. France, J. M. Bieman, and B. H. C. Cheng. Repository for Model Driven Development (ReMoDD). In *MoDELS Workshops'06*, pages 311–317, 2006. 34, 35

[22] Robert B. France and Bernhard Rumpe. Domain specific modeling. *Software and System Modeling*, 4(1):1–3, 2005. 15, 77

[23] E. Gamma, R. Helm, R. E. Johnson, and J.Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. 32, 47

[24] V. Di Giacomo, M. Felici, V. Meduri, D. Presenza, C. Riccucci, and A. Tedeschi. Using security and dependability patterns for reaction processes. In *Proceedings of the 2008 19th International Conference on Database and Expert Systems Application*, pages 315–319, Washington, DC, USA, 2008. IEEE Computer Society. 17, 32, 33

[25] R. Girardi and A. Neres Lindoso. An ontology-based knowledge base for the representation and reuse of software patterns. *ACM SIGSOFT Software Engineering Notes*, 31(1):1–6, 2006. 35

[26] J. Gray, J-P. Tolvanen, S. Kelly, A. Gokhale, S. Neema, and J. Sprinkle. *Domain-Specific Modeling*. Chapman & Hall/CRC, 2007. 15, 77

[27] B. Hamid. SEMCO Project (System and software Engineering for embedded systems applications with Multi-COncerns support). http://www.semcomdt.org. 19

[28] B. Hamid, N. Desnos, C. Grepet, and C. Jouvray. Model-based security and dependability patterns in RCES: the TERESA approach. In *1st International Workshop on Security and Dependability for Resource Constrained Embedded Systems (SD4RCES)*, 2010. 5

[29] B. Hamid, J. Geisel, A. Ziani, JM. Bruel, and J. Perez. Model-driven engineering for trusted embedded systems based on security and dependability patterns. In *SDL Forum*, pages 72–90, 2013. 6

[30] B. Hamid, J. Geisel, A. Ziani, and D. Gonzalez. Safety Lifecycle Development Process Modeling for Embedded Systems - Example of Railway Domain (regular paper). In *Software Engineering for Resilient Systems (SERENE),*

Pisa, Italy, 27/09/2012-28/09/2012, volume 7527 of *LNCS*, pages 63–75, http://www.springerlink.com, septembre 2012. Springer. 7

[31] B. Hamid, S.Gurgens, C. Jouvray, and N. Desnos. Enforcing S&D Pattern Design in RCES with Modeling and Formal Approaches. In Jon Whittle, editor, *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*, volume 6981, pages 319–333. Springer, octobre 2011. 125

[32] B. Hamid and A. Ziani. An Environment for Design Software and Hardware Aspects of Clock Synchronization and Communication in DRTES. In *IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing, EUC 2010*, pages 60–67. IEEE, 2010. 7

[33] B. Hamid, A. Ziani, and J. Geisel. Towards Tool Support for Pattern-Based Secure and Dependable Systems Development (regular paper). In *ACadeMics Tooling with Eclipse (ACME), Montpellier, France, 02/07/2013-02/07/2013*, http://portal.acm.org/dl.cfm, 2013. ACM DL. 5

[34] David Harel and Bernhard Rumpe. Modeling languages: Syntax, semantics and all that stuff part i: The basic stuff. Technical report, 2000. 15, 77

[35] T. Henzinger and J. Sifakis. The embedded systems design challenge. In *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*, pages 1–15, Ontario, Canada, August 2006. Springer. 1, 9

[36] T. Holmes, U. Zdun, and S. Dustdar. MORSE: A Model-Aware Service Environment, 2009. 35

[37] ISO. ISO 7498-2. . Information processing system - Open systems interconnections - Basic reference model - Part 2: Security architecture. Technical Report, 1989. 36

[38] ISO/IEC. ISO/IEC 13335-1. Information technology - Guidelines for the management of IT security - Part 1: Concepts and models for IT security . Technical Report, 1996. 36

[39] ISO/IEC. Standard: ISO/IEC 13335, 2004. 35

[40] F. Gomez C. Jouvray Y. Rouxel A. Perez J. L. Fernadez, R. Alonzo. Requirements engineering of trusted embedded systems. *INCOSE*, 2010. 13

[41] R.E. Johnson and M. Hfiz. Security patterns and their classification schemes. Technical Report, 2006. 36

[42] A. G. Kleppe. A language description is more than a metamodel. In *Fourth International Workshop on Software Language Engineering, Nashville, USA*, Grenoble, France, 2007. megaplanet.org. 15, 77

[43] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. 15

[44] C. Mayr, U. Zdun, and S. Dustdar. Reusable Architectural Decision Model for Model and Metadata Repositories. In *FMCO*, pages 1–20, 2008. 34

[45] J. McAffer, J-M. Lemieux, and C. Aniszczyk. *Eclipse Rich Client Platform*. Addison-Wesley Professional, 2nd edition, 2010. 19

[46] J. Miller and J. Mukerji. Mda guide version 1.0.1. Technical report, Object Management Group (OMG), 2003. 15

[47] Tammy Noergaard. *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Newnes, 2005. 11

[48] OASIS. EbXML: Oasis Registry Services Specification v2.5, 2003. 34

[49] OBEO. Acceleo. http://www.eclipse.org/acceleo/. 19

[50] OMG. MOF QVT Final Adopted Specification, 2005. 15

[51] OMG. UML Profile for Schedulability, Performance, and Time Specification. http://www.omg.org/technology/documents/formal/schedulability.htm, January 2005. 28

[52] OMG. Meta Object Facility (MOF) Core Specification Version 2.0. http://www.omg.org/cgi-bin/doc?formal/2006-01-01, 2006. 15

[53] OMG. A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems,Beta 2. http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf, June 2008. 20, 28

[54] OMG. OMG Systems Modeling Language (OMG SysML). http://www.omg.org/spec/SysML/1.1/, November 2008. 28, 29

[55] OMG. OMG Unified Modeling Language (OMG UML), Superstructure. http://www.omg.org/spec/UML/2.2/Superstructure, February 2009. 15

[56] OMG. OCL 2.2 Specification, February 2010. 15

[57] N. F. Neves P. E. Verissimo and M. P. Correia. Intrusion-tolerant architectures: Concepts and design. *Architecting Dependable Systems*, 2:33–36, 2003. 11

[58] D. Bruce Powel. *Real-time design patterns : robust scalable architecture for real-time systems*. The Addison-Wesley object technology series. Addison-Wesley, Boston, San Francisco, Paris, 2003. 126, 132

[59] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady. Security in embedded systems: Design challenges. *ACM Trans. Embed. Comput. Syst.*, 3(3):461–491, 2004. 1, 129

[60] Saluka R. S. R. Kodituwakku and P. Bertok. Pattern categories: A mathematical approach for organizing design patterns. In James Noble, editor, *Pattern Languages of Programs 2002. Revised papers from the Third Asia-Pacific Conference on Pattern Languages of Programs, (KoalaPLoP 2002)*, volume 13 of *CRPIT*, page 63, Melbourne, Australia, 2003. ACS. 35

[61] Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjodin. Toward model-based trade-off analysis of non-functional requirements. *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 142–149, 2012. 31

[62] SAE. Architecture Analysis & Design Language (AADL). http://www.sae.org/technical/standards/AS5506A, January 2009. 28, 29

[63] C. Sapia, M. Blaschka, and G. Höfling. GraMMi: Using a Standard Repository Management System to Build a Generic Graphical Modeling Tool. In *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8 - Volume 8*, HICSS '00, pages 8058–. IEEE Computer Society, 2000. 35

[64] D. Schmidt. Model-driven engineering. *in IEEE computer*, 39(2):41–47, 2006. 1, 14

[65] M. Schumacher. *Security Engineering with Patterns - Origins, Theoretical Models, and New Applications*, volume 2754 of *Lecture Notes in Computer Science*. Springer, 2003. 16, 20

[66] D. Serrano, A. Mana, and A-D Sotirious. Towards Precise and Certified Security Patterns. In *Proceedings of 2nd International Workshop on Secure systems methodologies using patterns (Spattern 2008)*, pages 287–291. IEEE Computer Society, September 2008. 33

[67] D. Steinberg, F. Budinsky, M. Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009. 18, 19, 77

[68] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley / ACM Press, Wesley, New York, 2002. 1, 40

[69] S. Taha, A. Radermacher, S. Gérard, and J-L. Dekeyser. An open framework for detailed hardware modeling. In *SIES*, pages 118–125, 2007. 29

[70] TERESA. Demonstration of teresa contributions to examples. Deliverable D6.3 – TERESA/WP6/D6.3, IST project ist-248410, January 2013. 104, 105

[71] TERESA. Repository structure specification. Deliverable D4.2 – TERESA/WP4/D4.2, IST project ist-248410, January 2013. 75

[72] TERESA. Repository v2. Deliverable D4.4 – TERESA/WP4/D4.4, IST project ist-248410, January 2013. 75

[73] TERESA. Specification of platform. Deliverable D6.1 – TERESA/WP6/D6.1, IST project ist-248410, January 2013. 105

[74] TERESA Consortium. TERESA Project (Trusted Computing Engineering for Resource Constrained Embedded Systems Applications). http://www.teresa-project.org/. 10

[75] F. Thomas, S. Gérard, J. Delatour, and F. Terrier. Software real-time resource modeling. In *FDL*, pages 231–236, 2007. 28

[76] Matthias Tichy, Daniela Schilling, and Holger Giese. Design of self-managing dependable systems with uml and fault tolerance patterns. In *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, WOSS '04, pages 105–109, New York, NY, USA, 2004. ACM. 17, 32, 33

[77] D Trowbridge, W Cunningham, M Evans, L Brader, and P Slater. Describing the enterprise architectural space. *MSDN*, 2004. 36

[78] Aneta Vulgarakis. Towards a resource-aware component model for embedded systems. In *Doctoral Symposium of 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2009)*. IEEE Computer Society Press, July 2009. 31

[79] Aneta Vulgarakis and Cristina Seceleanu. Embedded systems resources: Views on modeling and analysis. In *1st IEEE International Workshop On Component-Based Design Of Resource-Constrained Systems (CORCS 2008)*. IEEE CS, July 2008. 11, 31

[80] Z. Yan, R. M. Dijkman, and P. Grefen. Business process model repositories - framework and survey. *Information & Software Technology*, 54(4):380–395, 2012. 36

[81] J. Yoder and J. Barcalow. Architectural patterns for enabling application security. In *Conference on Pattern Languages of Programs (PLoP 1997)*, 1998. 16, 17, 32, 33

[82] N. Yoshioka, H. Washizaki, and K. Maruyama. A survey of security patterns. *Progress in Informatics*, (5):35–47, 2008. 18, 32, 33

[83] John A. Zachman. A framework for information systems architecture. *IBM Syst. J.*, 26:276–292, September 1987. 36

[84] A. Ziani and B. Hamid. Clock Synchronization Modeling in DRTES (regular paper). In *Hands-on Platforms and tools for model-based engineering of Embedded Systems (workshop at ECMFA 2010) (HoPES), Paris, 15/06/2010-16/06/2010*, pages 51–56, http://www-list.cea.fr, 2010. CEA LIST. 7

[85] A. Ziani, B. Hamid, and JM. Bruel. A model-driven engineering framework for fault tolerance in dependable embedded systems design. In *EUROMICRO-SEAA*, pages 166–169, 2012. 7

[86] A. Ziani, B. Hamid, J. Geisel, and JM. Bruel. A Model-based Repository of Security and Dependability Patterns for Trusted RCES (regular paper). In *IEEE International Conference on Information Reuse and Integration (IRI), San Francisco, CA. USA, 14/08/2013-16/08/2013*, http://www.ieee.org/, 2013. IEEE. 5

[87] A. Ziani, B. Hamid, and S. Trujillo. Towards a Unified Meta-model for Resources-Constrained Embedded Systems. In *37th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 485–492. IEEE, 2011. 6, 125

[88] R. Zurawski. *Embedded Systems*. CRC Press Inc, 2005. 1

Appendix A. Abbreviations

- ARABION: Pattern Editor
- DIPM : Domain Independent Pattern Model
- DSPM : Domain Specific Pattern Model
- GAYA : CDO-based Repository infrastructure
- GPRM : Generic Property Metamodel
- MATHO : Resource Editor
- NFP : Non Functional Properties
- PBSE: Pattern-based System and software Engineering
- RCES : Resource Constrained Embedded Systems
- SARM : System and software Artifacts Repository Model
- SEMCO : System & Software Engineering with Multi-COncerns
- SEMCOML : SEMCO Modeling Language
- SEMCOMDT : SEMCO Model Development Tools
- SEPM : System and software Pattern Metamodel
- SERM : System & Software Engineering Resource Metamodel
- S&D : Security and Dependability
- TERESA : Trusted computing Engineering for Resource constrained Embedded Systems Applications
- TIQUEO : Property Editor

Appendix B. Patterns Description

8.2.1 Watchdog

The Watchdog is a lightweight and inexpensive Safety Related pattern [54]. The watchdog checks whether the monitored application or a particular (safety) process of the application is running into its time base or executed in the correct order. The Watchdog is commonly used in real-time and dependable applications, in order to ensure that one or more critical time requirements are met by the application. It can also be combined with additional monitoring patterns in order to increase diagnostic coverage of the safety system.

The Watchdog pattern based on IEC-61508 safety standard is a temporal and logical program sequence executing monitor with the following template:

- Type: Hardware / software / VHDL
- Aim: To detect time-out, a wrong program sequence executing or a fault in the process' clock.
- Context: Development of a fail-safe embedded system (IEC-61508, EN-5012X, etc.)
- Problem: How to detect a time-out, wrong program sequence executing or a fault in the process clock
- Solution: The watchdog is refreshed by the safety application (indicating correct program sequence) and the watchdog triggers a hardware reset if timing constraints are not met.
- Safety recommendation: IEC-61508 [SIL1 (HR), SIL2 (HR), SIL3 (HR), SIL4 (HR)]

8.2.2 Black Channel - Safety Communication Layer

Distributed safety systems require (safe) data communication among distributed safety functions. Data communication is required to be safe and this leads to two possible approaches [28]:

- "White channel": Usage of a safety communication channel designed, implemented and validated according to IEC-61508 series and IEC-61784-3 [50] or IEC-62280 [55] series. That means usage of a certified safety communication channel such as TTEthernet (TTE) [56].
- "Black channel": Usage of a communication channel not designed or validated according to IEC-61508 series where safety measures shall be implemented either in the safety functions or in interfaces with the communication layer in accordance with IEC-61784-3 or IEC-62280 series. Where the latter is called a Safety Communication Layer (SCL).

As shown in the following figure, the Safety Communication Layer (SCL) IEC-61784 / IEC-62280 is an application level service on top of a non-safety related communication stack ('comms') that enables "safe" data exchange between safety functions. It must be developed with a life cycle equivalent to the highest safety level (SIL) in the application and it requires the detection of all possible communication errors as described by IEC-61784 / IEC-62280: corruption, message incorrect order, message outside temporal requirements, message lost, message duplicated, etc.

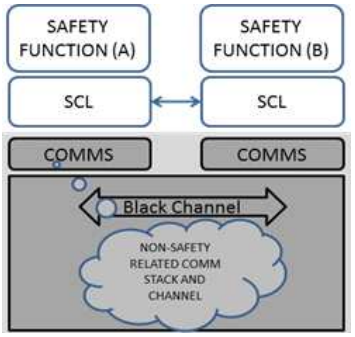

Figure 8.2: The (simplified) Black Channel Diagram

The black channel - safety communication layer pattern profile is described below:

- Type: Hardware / software / VHDL
- Aim: To provide safe data communication among distributed functions communicating with a 'black channel'

- Context: Development of a safety distributed embedded system interconnected with a 'black channel' communication (IEC-61508, EN-5012X, etc.)
- Problem: How to detect all possible error modes of a 'black channel' communication
- Solution: Usage of a Safety Communication Layer (SCL) that meets IEC-61784 (generic) and / or IEC-62280 (railway) and developed according to IEC-61508
- Safety recommendation: N/A

8.2.3 Secure Communication Layer

Distributed components of a distributed system share their information and data through wired or wireless connections that can be the objective of an external attacker, which can disturb the operation of the system and even compromise system availability and safety [57]. The secure communication layer enables a secure data exchange between components over a non-secure communication channel (e.g. Ethernet), assuring the data integrity and authenticity of the sender.

Figure 8.2 shows a brief schema of what is expected from this secure communication layer pattern. Messages sent among distributed system functions (components) shall arrive from authorized sender(s) and without data modification. If any attacker sends a message or modifies an existing one, this message should be discarded by the receiver security communication layer and destination function might be informed about this action.

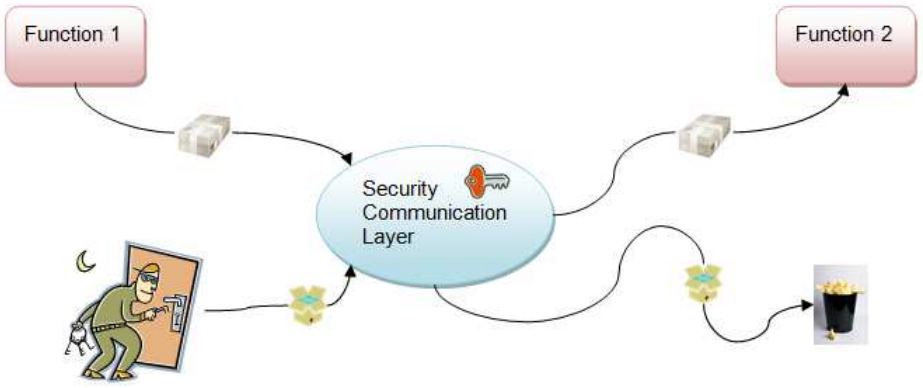

Figure 8.3: Secure communication layer schema

The Secure Communication Layer (SeCL) pattern profile is described below:

- Type: Software / VHDL

- Aim: To provide secure data communication among distributed functions communicating through a non-secure channel.
- Context: Development of a safety distributed embedded system.
- Problem: An external attacker can impersonate a system component and compromise the safety by sending erroneous information.
- Solution: Using a secure communication that codifies and authenticates the sender of the information as a valid system component.
- Safety recommendation: N/A

8.2.4 Triple Modular Redundancy (TMR)

The Triple Modular Redundancy (TMR) is a fault-tolerant redundant architecture in which three computational channels perform a safety computation and the result is voted to produce a single safe output. Based on the IEC-61508-2 safety standard (Table 3) a TMR architecture is a reasonable solution to reach SIL4 levels with a single hardware fault tolerance (HFT=1). This means that in case one computation channel fails due to the presence of a random hardware fault (no systematic fault), the remaining two computation channels and voting system can still safely execute the safety system.

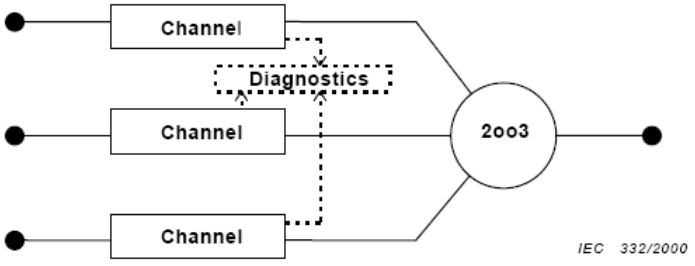

Figure 8.4: TMR (2oo3) physical block diagram as described by IEC-61508-7 [58]

The Triple Module Redundancy (TMR) Pattern template is described below.

- Type: System hardware level
- Aim: Support at least single random hardware fault-tolerance
- Context:

  - Generic: Development of an IEC-61508 based fail-safe embedded system that shall provide a single random hardware fault-tolerance (HFT=1, IEC-61508-2 Table 3).
  - Railway: Development of a EN-50126 based fail-safe embedded system, with a "composite fail-safety" technique (EN-50129 B.3.1 Effects of single faults, Table E.4)

- Problem: How to deal with random faults and single-point of failure (HFT=1, IEC-61508) in order to increase the safety of the system to required SIL level

- Solution: Triple Modular Redundancy (TMR) that meets IEC-61508-2 (HFT=1, IEC-61508-2 Table 3); for the railway domain it should also be compliant with "composite fail-safety" technique (EN-50129 B.3.1 Effects of single faults, Table E.4)

- Recommendation: Reasonable solution to reach SIL4 levels with HFT=1 while supporting an achievable Diagnostic Coverage (>=99%). Exceptionally if appropriate measures are taken, an HFT=2 can be achieved with Diagnostic Coverage > 90%.

- Related: At later design stages, this system level pattern leads to software type patterns such as data agreement, majority voter, reciprocal monitoring, etc.

8.2.5 Majority Voter

A majority voter according to IEC-61508-2/-7 [59, 60] is a safety technique that provides a safe output based on the majority principle (M out of N. e.g. 2oo3), masking failures in one of at least three hardware channels (TMR, see Figure 109). The voter itself might be diagnosed using self-monitoring technology or externally tested.

The majority voter pattern template is described below.

- Type: Software / Hardware / VHDL

- Origin: Redundancy patterns (e.g. TMR) that must reach an agreement based on majority voter

- Context:

  - Generic: Development of an IEC-61508 based fault-tolerant software for a highly safety-critical system where redundant software is executed.
  - Railway: Development of an EN-50128 based fault-tolerant software for a highly safety-critical system where redundant software is executed (e.g. "composite fail-safety" technique based system)

- Problem: How to reach a safe agreement among data used / produced by replicated safety software
- Solution: Majority voting that provides a safe output based on the majority principle (M out of N. e.g. 2oo3), masking failures in one of at least three hardware channels (TMR)
- Recommendation: Majority voter provides a high diagnostic coverage for electronic components, electrical components and processing units (IEC-61508-2 Table A-2, Table A.3 and Table A.4). It is also required in TMR based safety applications.
- Related: TMR

8.2.6 Reciprocal Monitoring

The Reciprocal Monitoring pattern is a monitoring and checking pattern between N data providers, also known as "monitored redundancy" (IEC-61508-7 [60]). If one of the providers is sending an erroneous data stream the other entities will detect and accuse it of having a fault.

- Type: Software / Hardware / VHDL
- Origin: Redundancy patterns (e.g. TMR)
- Requires: Majority voter pattern or equivalent (compare redundant data and detect failure)
- Aim: IEC-61508-7 (Monitored redundancy) "To detect failure, by providing several functional units, by monitoring the behaviour of each of these to detect failures, and by initiating a transition to a safe condition if any discrepancy in behaviour is detected"
- Context:

  - Generic: Development of an IEC-61508-7 based fault-tolerant software for a highly safety-critical system where the interchanged data among redundant software is monitored in order to detect errors / failures
  - Railway: Development of an EN-50128 based fault-tolerant software for a highly safety-critical system where the interchanged data among redundant software is monitored in order to detect errors / failures (e.g. "composite fail-safety" technique based system)

- Problem: How to detect a failure among replicated software
- Solution: Periodical information interchange among redundant software is monitored by each replica in order to detect errors / failures
- Related: TMR at earlier design stages, safety concept and system architecture. Majority voter at software architecture and software detailed design phase.

8.2.7 Data Agreement

The Data Agreement Pattern [57] is a conjunction of the Majority Voter and Reciprocal Monitoring Pattern. This domain specific pattern includes both of them in order to obtain a unique safe output and the diagnosis of the three computation units that compound the pattern.

- Type: Software / Hardware / VHDL
- Origin: Redundancy patterns (e.g. TMR)
- Requires: Majority voter and reciprocal monitoring pattern or equivalent
- Aim: Reach a system level agreement on input / output data used by redundant software
- Context:
  - Generic: Development of an IEC-61508 based fault-tolerant software for a highly safety-critical system where data (input / output) is managed and used by redundant software must be agreed in order to provide safe outputs and in order to detect failures
  - Railway: Development of an EN-50128 based fault-tolerant software for a highly safety-critical system where data (input / output) is managed and

used by redundant software must be agreed in order to provide safe outputs and in order to detect failures (e.g. "composite fail-safety" technique based system)

- Problem: How to reach an agreement among replicated software of the data to be used (input / output) and to detect a failure if no agreement is reached
- Solution: Data agreement among replicated software of the data to be used by means of information interchange, majority voting and reciprocal monitoring.
- Related: TMR at earlier design stages, safety concept and system architecture. Majority voter and reciprocal monitoring at software architecture and software detailed design phase.