A Model-Driven Approach for Developing a Model Repository: Methodology and Tool Support

Brahim Hamid

A model-driven approach for developing a model repository: Methodology and tool support

Brahim Hamid *

IRIT, University of Toulosue, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France

HIGHLIGHTS

- Repository design: we describe a design framework and infrastructure to support the model-based repository lifecycle.
- Modeling: we propose a modeling language to specify a model-based repository independent from end-development applications and execution platforms.
- Tooling: we describe an operational architecture for development tools to support the proposed approach.
- Validation: we apply the approach to a resource-constrained embedded system (RCES) in the context of the TERESA project : Railway Systems.

ABSTRACT

Several development approaches have been proposed to cope with the increasing complexity of embedded system design. The most widely used approaches are those using models as the main artifacts to be constructed and maintained. The desired role of models is to ease, systematize and standardize the approach to the construction of software-based systems. To enforce reuse and interconnect the process of model specification and system development with models, we promote a model-based approach coupled with a model repository. In this paper, we propose a model-driven engineering methodological approach for the development of a model repository and an operational architecture for development tools. In addition, we provide evidence of the benefits and feasibility of our approach by reporting on a preliminary prototype that provides a model-based repository of security and dependability (S&D) pattern models. Finally, we apply the proposed approach in practice to a use case from the railway domain with strong S&D requirements.

Keywords:
Modeling artifact
Repository
Meta-model
Model-driven engineering
Security & dependability patterns

1. Introduction

1.1. Motivation and background

Designers and developers of next-generation software systems are faced with the exponential challenge of managing the continuously increasing requirements of such systems. For instance, non-functional requirements, such as security and dependability (S&D) [1], have become increasingly important as well as increasingly difficult to achieve. As a result, new recommendations should be considered to build novel methods capable of handling the complexity and reducing the cost of the development of these systems. The specification and packaging of software modeling artifacts can provide an efficient way to address these problems, improving the industrial efficiency, fostering technology *reuse* across domains (reuse of models at different levels), and thus reducing the amount of effort and time needed to design a complex system.

Repositories of modeling artifacts have gained increasingly attention recently to enforce reuse in software engineering. In fact, repository-centric development processes are often adopted in software/system development, such as architecture- or pattern-centric development processes.

According to Bernstein and Dayal [2], a repository is a shared database of information on engineered artifacts. These researchers state 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 our work, we go one step further by proposing a model-based repository to support the specifications, definitions and packaging of a set of modeling artifacts to assist the developers of trusted applications for embedded systems. Closely related to our vision is the approach for specifying, designing and implementing a reuse repository presented in [3].

* Fax: +33 (0)5 6150 4173.
E-mail address: Hamid@irit.fr.

In this paper, we propose a model-driven engineering (MDE) approach to produce a repository of modeling artifacts and an operational implementation in the context of the FP7 TERESA project [4]. Modeling artifacts derived from and associated with domain-specific models will help developers to integrate in-development application building blocks with pre-defined modeling artifact building blocks. Additionally, various services dedicated to repository features will be developed in this task. The goal is to integrate features together via model-based repository engineering coupled with MDE technology, thus allowing the reuse of model building blocks from the repository to be leveraged.

1.2. Development context

In recent years, there has been a paradigm shift in terms of design with the combination of multiple software engineering paradigms; namely, MDE [5], component-based software engineering (CBSE) [6] and software reuse [7]. Such a paradigm shift is changing the way in which systems are currently developed and reducing development time significantly.

In our work, we promote a new discipline for system and software engineering that uses modeling artifacts as its first-class citizens. We build on a theory and novel approach called SEMCO (System and software Engineering with Multi-COncerns support) [8], which is based on an integrated repository of modeling artifacts working as a group, each relevant to a key concern. The associated framework provides methods, modeling languages and a tool-chain to support two categories of users: "reuse" producers and "reuse" consumers. The former category encompasses developers of artifacts to be stored in the repository, whereas the latter concerns developers who reuse existing artifacts from the repository.

Models are used to denote abstract representations of computing systems. Specifically, we need models to represent software architectures and software platforms to test, simulate and validate the proposed solutions. MDE promotes models as first-class elements. A model can be represented at different levels of abstraction, and the MDE vision is based on (1) the meta-modeling techniques to describe these models and (2) the mechanisms to specify the relations between them. Model exchange, as well as the transformation/refinement relation between two models, is at the core of the MDE methodology. In software engineering, domain-specific modeling (DSM) [9] is a methodology that uses models to specify applications within a particular domain. There are several DSM environments, one being the open-source Eclipse Modeling Framework (EMF) [10]. The EMF provides an implementation of essential meta object facility (EMOF), a subset of the MOF [11], called Ecore.[1] However, our vision is not limited to the EMF platform. The EMF offers a set of tools to specify metamodels in Ecore and to generate other representations of them. Query view transformation (QVT) [11] is a standard to specify model transformations between metamodels conforming to the MOF in a formal manner. In our vision, the SEMCO framework is based on a federation of domain-specific modeling languages (DSMLs) built on an integrated repository of modeling artifacts.

The industrial context of our work is the question of how to account for several constraints, mainly those related to S&D that are not satisfied by the well-known and widely used technology for building applications for resource-constrained embedded systems (RCESs). Such systems have many common characteristics, including real-time, temperature, memory, computational processing power and/or energy consumption constraints, as well as security, dependability and efficiency requirements. The computing resources of RCESs, e.g., memory, tasks, and buffers, are generally statically determined. Thus, the generation of RCESs involves specific software building processes. Furthermore, many RCESs have assurance requirements, ranging from strong levels involving certification (e.g., DO178 and IEC-61508 for the development of safety-relevant embedded systems) to lighter levels based on industry practices.

These requirements introduce conflicts for the three main factors that determine the owner-ship cost of applications: the cost of production, the cost of engineering and the cost of maintenance. In other words, systems with high dependability requirements for which the security level must be demonstrated and certified nearly exclusively use technical solutions that are strongly oriented by the application domains. Applications based on these solutions are dedicated by definition; they are not typically portable between different execution platforms and require specific engineering processes. These specificities greatly increase the cost of development in the different phases of their lifecycle. For instance, the integration of S&D features requires the availability of both application domain-specific knowledge and S&D expertise. Hence, capturing and providing this expertise via modeling artifacts can enhance the development of embedded systems.

1.3. Intended contribution

The envisioned repository framework will be comprised of two main pillars: solid theory and proven principles. The first pillar will offer an integrated conceptual design for the specification of the repository structure; the second pillar will offer a set of concrete and coherent techniques to generate a model-based repository. We will provide evidence of its benefits and applicability through an example of a representative industrial case from the TERESA project: safe4Rail application. The basis formulation of the approach presented in this article has been previously published in a research paper at the 4th International Conference on Model and Data Engineering (MEDI'14) [12]. This work extends ideas described in this earlier paper and presents a holistic approach for the design, implementation and management of a model-based repository of modeling artifacts. Specifically, we provide a more comprehensive and complete description of our approach. The work presented in this paper has the following aspects:

- Repository design: a design framework and infrastructure to support the model-based repository lifecycle.
- Modeling: we propose a modeling language to specify a model-based repository independent from end-development applications and execution platforms.
- Tooling: we propose a connected data objects (CDO)[2]-based implementation of the repository and a set of EMF tree-based editors to create patterns and the required libraries.
- Validation: applied in practice to a resource-constrained embedded system (RCES) with strong S&D requirements in the context of the TERESA project [4,13]: Railway Systems.

1.4. Organization of the contribution

The remainder of this paper is structured as follows. In Section 2, we highlight the overall approach for designing and developing a model-based repository of modeling artifacts. Section 3 presents the basic concepts as the basis for the definition of a modeling language to support the design of the repository structure and

[1] Ecore is a meta-metamodel.

[2] http://www.eclipse.org/cdo/.

its interfaces. In Section 4, we describe the proposed methodology for designing, implementing and exploiting the repository of modeling artifacts. Section 5 describes the architecture of the tool suite and an example of an implementation of a repository. Section 6 describes a modeling framework for the modeling artifact-based system and software engineering around a model-based repository, as well as the usage of the defined modeling framework in the context of the FP7 TERESA project through the railway case study. In Section 7, we present a review of the most important related work. Finally, Section 8 concludes and presents future work directions. We investigate various open issues, mainly the issues of generalization and implementation, including the usability of the proposed modeling framework.

2. Approach

In this section, we provide a description of the global vision of a model-based repository system as a living structure of models for software systems engineering. Fig. 1 highlights the architecture of the model-based repository system lifecycle, decomposed into components, as in [3]. The main components of the model-based repository are (1) Design, (2) Infrastructure, (3) Development for reuse, (4) Management and (5) Development by reuse. In this section, we describe each component and provide indications as to how these components work in conjunction to cover all aspects of the model-based repository lifecycle.

In the next section, we present the specification language for the repository and its interfaces, and in Section 5, we detail an implementation of a model-based repository of S&D patterns.

2.1. Design and specification

The core of the framework is the definition of the model of the repository, including the structure and interfaces, and a set of specification languages for the modeling artifacts to be stored in this repository. In addition, we specify views on the repository for access according to its interfaces, its organization and the needs of the targeted system engineering process.

2.2. Infrastructure for the repository software

Once this specification language has been defined, it is possible to develop a repository. The repository software is proposed as a platform where the modeling artifact specifications and instances are stored and managed. Another element of the infrastructure component is the interfaces, which are responsible for supporting repository interactions. Finally, the infrastructure provides tools for the installation, deployment and configuration of the repository software with respect to the configuration of the accompanying development tools that are installed in the user development environment.

2.3. Development for reuse environment

The development environment associated with the repository is composed of design and validation tools that define modeling artifacts and generate reports and documentation to populate the repository. Moreover, these tools should implement the appropriate interfaces to publish the results in the repository.

2.4. Management environment

For the repository management, to be used by the repository manager, we provide a set of facilities for the repository organization that allow for the enhancement of its usage. We also provide basic features, such as user, domain and artifact management. Moreover, we provide features to support the management of the relationships among artifact specifications and between artifact specifications and their complementary models. The tools belonging to this component should also implement appropriate interfaces.

2.5. Development by reuse environment

Once the repository[3] is available, it serves an underlying system engineering process through access tools for reuse. For a system engineer accessing the repository, this component provides a set of facilities to help in selecting appropriate modeling artifacts, including keyword search, lifecycle stage search and relationship types. The access tools include features for exportation and instantiation as dialogues targeting domain-specific development environments. Moreover, the tool includes dependency-checking mechanisms.

3. Repository metamodel

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 to use them in engineering processes. We start with a set of definitions and concepts that might prove useful in understanding our approach.

Definition 1 (*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 system engineering processes. The modeling artifact may be classified in accordance with engineering process levels.

Adapting the definition of pattern language given by Christopher Alexander [14], we define the following:

Definition 2 (*Modeling Artifact System*)**.** A modeling artifact language 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 bijective.

3.1. System and software artifact repository model (SARM) specification

The specification of the structure of the repository is based on the organization of its content, mainly the modeling artifacts and 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 is dedicated to storing and managing data in the form of *Compartments*, and the second regards the *Interfaces* for publishing and retrieving modeling artifacts and for managing interactions between users and the repository. The principal classes of the metamodel are described with the Ecore notations in Fig. 2. The following part depicts the meaning of principal concepts used to specify the repository in detail:

[3] The repository system populated with artifacts.

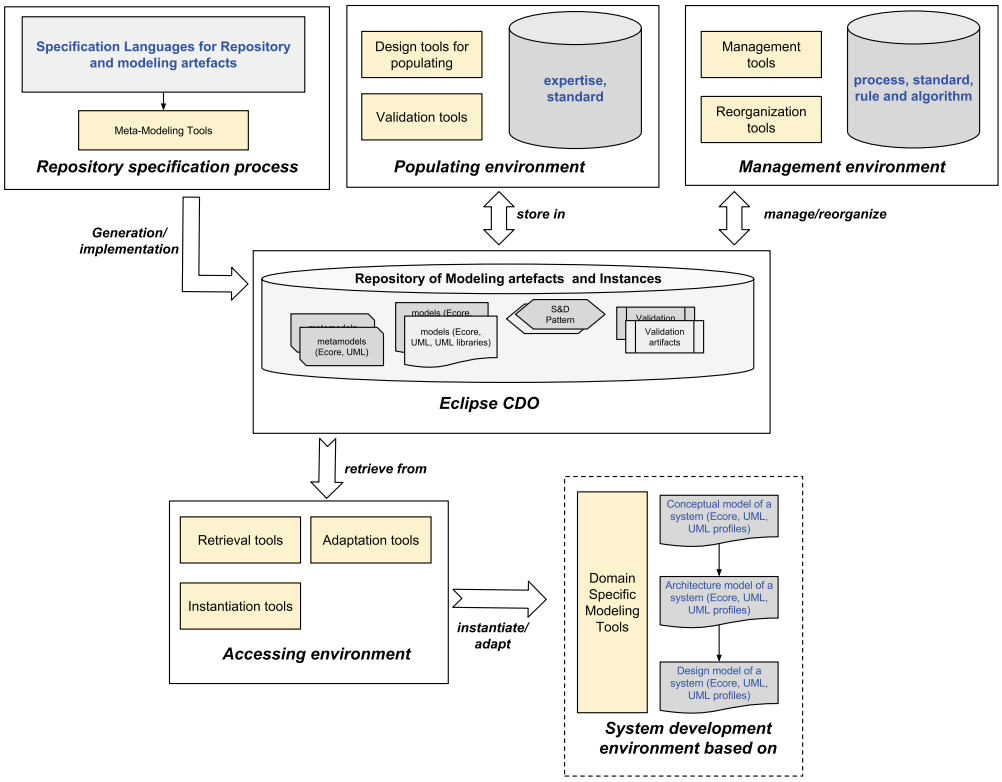

Fig. 1. Proposed framework for the repository system lifecycle.

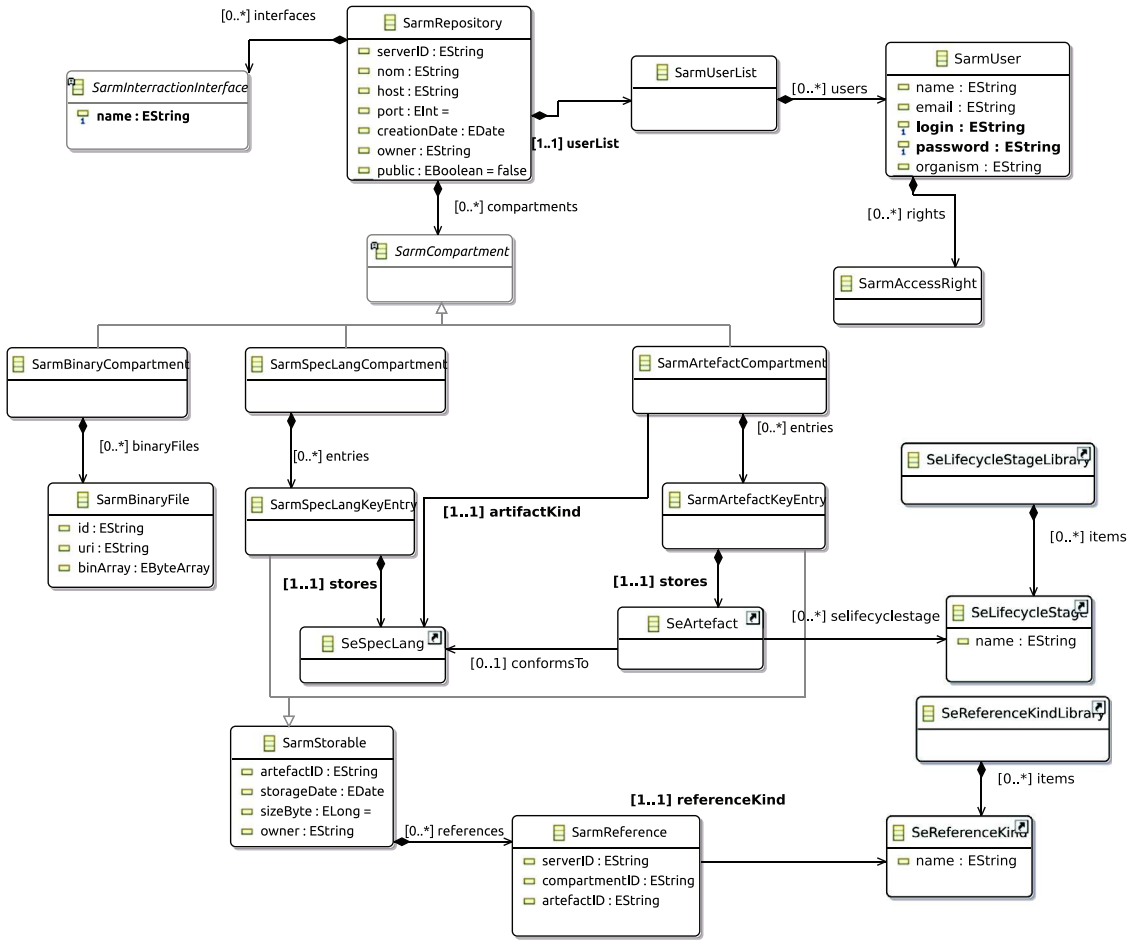

Fig. 2. Repository specification metamodel (SARM).

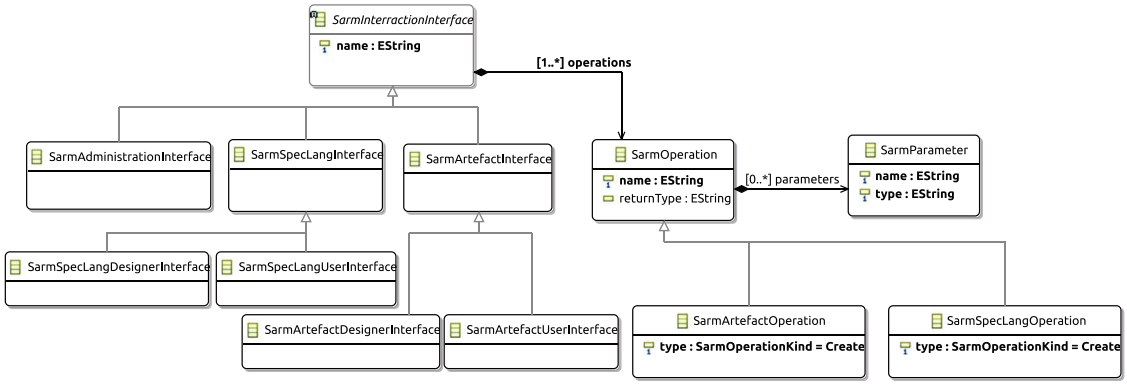

Fig. 3. Repository interface specification metamodel.

- SarmRepository. The core element used to define a repository.
- SarmCompartment. Used for categorizing the stored data. We have identified two main types of compartments.
  - SarmSpecLangCompartment. Used to store the specification languages (SeSpecLang) of the modeling artifacts (SeArtefact).
  - SarmArtefactCompartment. Used to store the modeling artifacts. To simplify the identification of a modeling artifact regarding the software development stage in which it is involved, an SeArtefact has a lifecycleStage typed with an external model library SeLifecycleStage.
- SarmStorable. Used to define a set of characteristics of the model-based repository content, mainly those related to storage. We can define: *artefactURI*, *storageDate*, *sizeByte*, and so on.
  In addition, it contains a set of references (SarmReference) to describe the different links with the other artifacts. The set of possible links is defined through an external model library SeReferenceKind.
- *SarmSpecLangKeyEntry*. The key entry to point towards a specification language model in the repository.
- *SarmArtefactKeyEntry*. The key entry to point towards a modeling artifact specification in the repository.
- *SarmAccesRight*. Used to define the characteristics regarding the access right to the repository and its content.
- *SarmUser*. Used to define the user profile.
- *SarmUserList*. Used to store the list of users in the repository.

3.2. Repository interface specification

For interaction purposes, the repository exposes its content through a set of interfaces (SarmInteractionInterface), as shown in Fig. 3. The meaning of the proposed concepts is presented in the following:

- *SarmAdministrationInterface*. Manages the repository.
- *SarmSpecLangDesignerInterface*. Offers a set of operations, including connection/disconnection to the repository and population of the repository with metamodels.
- *SarmSpecLangUserInterface*. Offers a set of operations, mainly including connection/disconnection to the repository and search/selection of the specification languages.
- *SarmArtifactDesignerInterface*. Offers a set of operations, including connection/disconnection to the repository and population of the repository with artifacts.
- *SarmArtefactUserInterface*. Offers a set of operations, mainly including connection/disconnection to the repository and search/selection of the modeling artifacts.

4. Methodology

In this section, we describe a methodological approach to describe the entire cycle comprising the creation of a flexible repository of modeling artifacts and managing the models in that repository, as shown in Fig. 4. We first describe the model-based repository building process (Section 4.2) and then present a description of the model-based repository usage process (Section 4.3). The first process describes the steps to be followed by the metamodelers, modelers and implementers of a model-based repository. The second process describes the steps to be followed by the modelers of modeling artifacts for reuse and the developers of software systems by reuse. For illustration purposes, in the remainder of this paper, we focus on the repository of S&D patterns, which acts as a specific demonstration for the TERESA resource-constrained embedded systems, called *TeresaRepository*.

The following sections introduce the example of *TeresaRepository* and describe in detail the process to be followed by the repository developers, including the designers of the metamodels of the artifacts and the modelers of these artifacts. The process describes the entire cycle, including the creation of the artifacts' metamodels, the instantiation of the repository metamodel, the instantiation of these metamodels as modeling artifacts for populating the repository, the management of the repository, and an overview on how the resulting repository software will support the system engineering process.

4.1. An S&D pattern repository

In the context of the TERESA project, we consider three types of modeling artifacts: S&D patterns, S&D property models and resource property models. In this vision, the S&D pattern, derived from (or associated with) domain-specific models, aims to help the system engineer integrate application S&D building blocks. Now, we briefly describe the modeling languages used to specify these artifacts. For more details on property modeling language and pattern modeling language, the reader is referred to [15,16], respectively.

4.1.1. Generic property modeling language

The Generic PRoperty Metamodel (GPRM) [15] is a metamodel defining a new formalism (i.e., language) for describing property libraries, including units, types and property categories. For instance, S&D attributes [17], such as authenticity, confidentiality and availability, are defined as categories. These categories require a set of measures types (e.g., degree, metrics) and units (e.g., Boolean, float). For this purpose, we instantiate the appropriate type library and its corresponding unit library. These models are

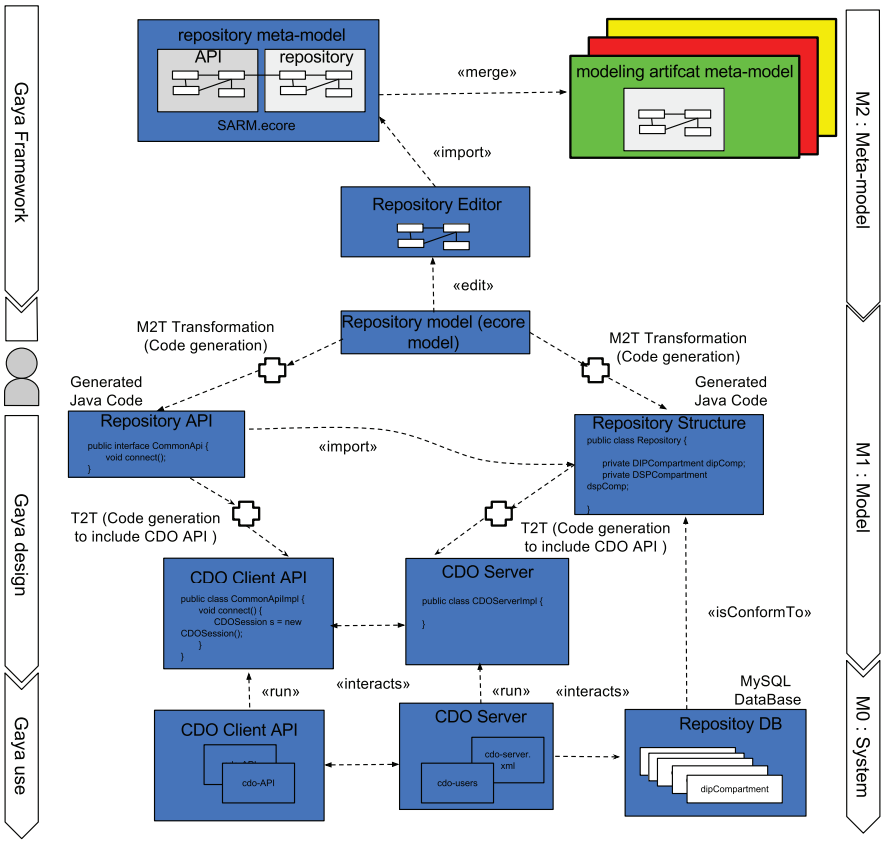

Fig. 4. Overview of the model-based repository building process.

used as external model libraries to type the properties of the patterns. During the editing of the pattern, we define the properties and constraints using these libraries. The principal classes of the GPRM are described with Ecore notations in Fig. 5. Their meanings are explained in more detail in the following paragraphs.

- *GprmProperty*. A property is a basic attribute shared by all members of an artifact (e.g., pattern, resource). As we shall see, it will be used to define pattern properties (see SEPM metamodels). The property is defined by its category, type and value.

  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 group all of 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, and 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 of resource properties, and *Authenticity* is a category of S&D properties.
- *GprmResourceCategory*. A resource category is a classification for resources. Its role is to group all of 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 energy consumption. However, other categories may be defined, such as peripheral, sensor, and actuator. A category may also be built based on existing categories by specialization.

4.1.2. Pattern specification metamodel

The System and Software Engineering Pattern Metamodel (SEPM) [16] is a metamodel defining a new formalism for describing S&D patterns and constitutes the base of our pattern modeling language. Here, we consider patterns as sub-systems that expose services (via interfaces) and manage S&D and resource properties (via features), yielding a unified method of capturing meta-information related to a pattern and its context of use. Fig. 6 describes the principal concepts of the SEPM metamodel with the Ecore notations. Their meanings are explained in more detail in the following paragraphs.

- *SepmPattern*. This block represents a security pattern as a subsystem describing a solution for a particular recurring security design problem that arises in a specific design context. A *SepmPattern* defines its behavior in terms of the provided and required interfaces.
- *Interface*. A *SepmPattern* interacts with its environment via *Interfaces*. We consider two types of interfaces:
  - *SepmExternalInterface*. Allows for the implementation of interaction with regard to the integration of a pattern into an application model or the composition of patterns.
  - *SepmTechnicalInterface*. Allows for the implementation of interaction with security primitives and protocols, such as encryption, and specialization for specific underlying software and/or hardware platforms, mainly during the deployment activity.
- *SepmProperty*. A *GprmProperty* denoting a particular characteristic of a pattern related to the concern it is considering and dedicated to capture its intent in a certain manner (e.g., S&D properties).

Example. We illustrate the use of the SEPM to specify a pattern with the example of a secure communication pattern based on the

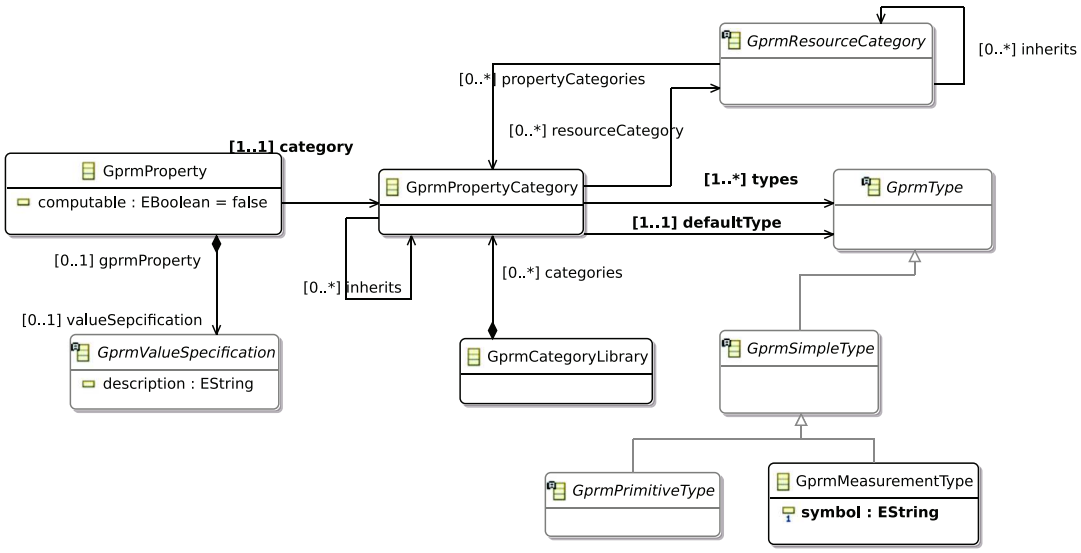

Fig. 5. The (simplified) GPRM.

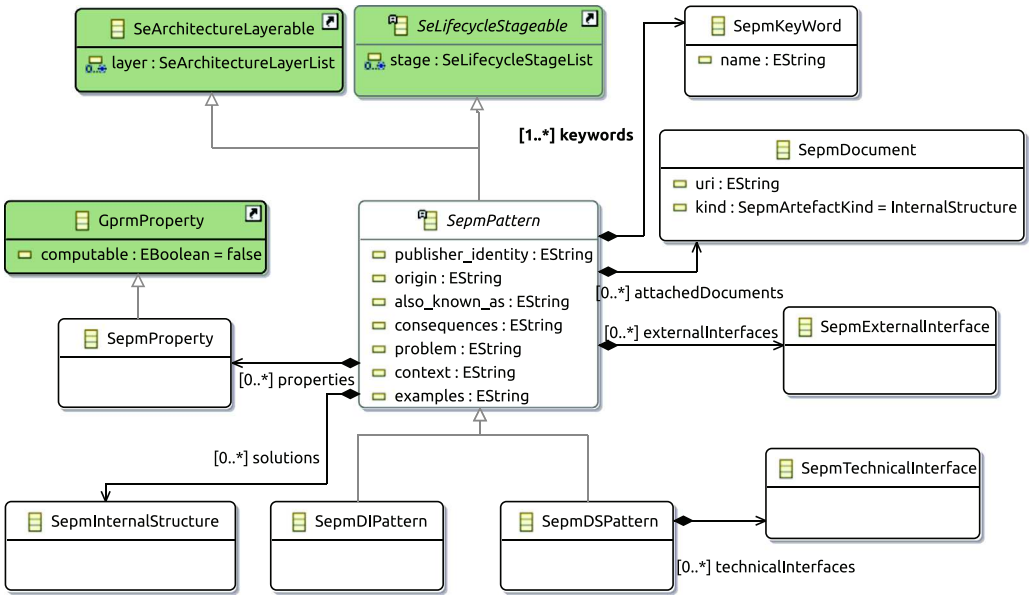

Fig. 6. The (simplified) SEPM.

SSL[4] mechanism. Here, we specify an S&D property: "authenticity of sender and receiver". We use a previously defined category from the S&D category library to type the category of this property: *Authenticity*. Moreover, we identify various resource properties, such as "CPU resource time for encryption" and "CPU resource time for authentication", which belong to the category *CPUTime*, and "extra energy cost for encryption" and "extra energy cost for authentication", which belong to the category *PowerConsumption*.

4.2. Steps for a model-based repository building process

Once we have developed the conceptual model of the repository (Section 3), we can create the modeling languages to specify its content and the appropriate tools to support the entire cycle of system engineering around a model-based repository of modeling artifacts. As we can see, the implementation steps were performed transversely to the other steps of the proposed methodology. The repository building process, as visualized in Fig. 7, consists of the following steps:

Create the artifacts' metamodel. Specify the metamodel of each artifact to be stored in the repository, as shown in the top part of Fig. 4. For instance, the GPRM and SEPM are created and stored as Ecore models (Figs. 5 and 6, respectively).

Specify model libraries for artifact classification. After the modeling artifacts' metamodels are specified, classification models are added. At each stage of the system engineering development process, the appropriate modeling artifacts to use are identified by classifying them. In our context, we use the pattern classification of Riehle and Buschmann [18,19], which is (1) *System Patterns*, *Architectural Patterns*, *Design Patterns* and *Implementation Patterns*, to create the model library SeLifecycleStage.

Specify model libraries for the relationships between artifacts. After the modeling artifacts' metamodels are specified, relationship models are added. At each stage (phase) *n* of the system engineering development process, the modeling artifacts previously

[4] The TLS Protocol Version 1.2. rfc5246, 2008.

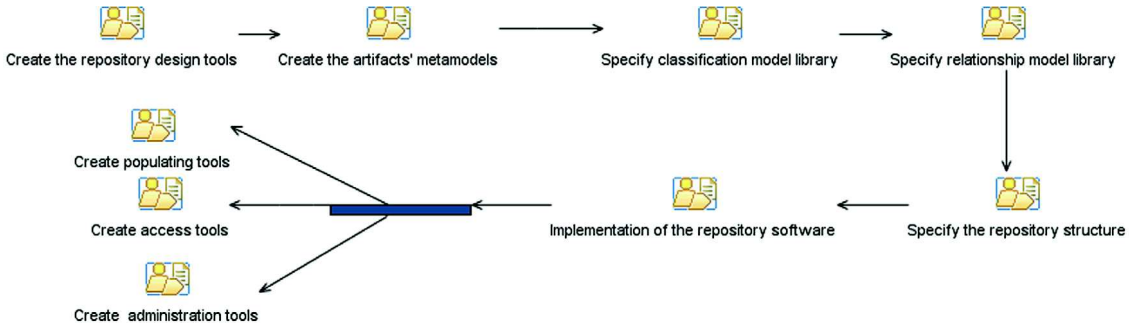

Fig. 7. The model-based repository building process.

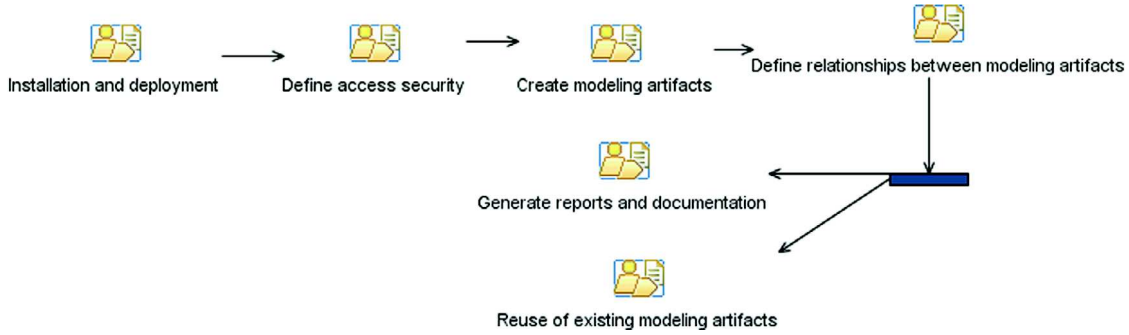

Fig. 8. The deployment and configuration process.

identified in stage $(n-1)$ will be used in the selection activity of the current phase. For instance, a pattern may be linked with other patterns and associated with property models using a predefined set of reference types, for example *refines*, *specializes*, and *uses*. Here, we create the SeReferenceKind model library to support the specification of relationships across artifacts.

Create tools to support the repository modeling process. Write editors for the specification of the repository structure and interfaces.

Specify the repository structure. Once a repository design tool is created, the repository model can be specified using the metamodels and model libraries. The model of the repository is an instance of the SARM metamodel, comprising the creation of metamodels' compartments, the artifacts' compartments, and the users' lists. The structure of the repository and its interfaces are then available to modelers to populate and manage the repository (as seen in the middle part of Fig. 4). In our example, we define *TeresaRepository* as an instance of the SarmRepository: a model-based repository of S&D patterns and their related property models. To implement S&D pattern models and property models, we use a *MetamodelCompartment* as an instance of the SarmSpecLangCompartment, which has two instances of SarmSpecLangKeyEntry to store the pattern modeling language and property modeling language. We also define a set of compartments to store the artifacts. In addition to the repository structure, we define the model of interfaces (APIs) to exhibit the content of the repository and its management.

Implementation of the repository software. Once the repository model has been specified, the repository software can be implemented using code generation techniques. The resulting models are used as input for the model transformations to generate the repository structure and interface (API) software implementations targeting a specific technological platform, such as CDO (see the middle part of Fig. 4). Additionally, scripts to perform the installation and deployment of the resulted repository software system are specified. The next step after creating the repository structure and interfaces, they are connected to populating, accessing and management tools.

Create tools to support the repository interactions. After defining the metamodels of artifacts and the specification and implementation of the repository structure and interfaces, repository interaction tools are developed. Each modeling artifact is associated with a set of editors and helpers to deposit and retrieve artifacts to/from the repository. Moreover, a set of management perspectives within the repository are proposed.

- *Create tools to support the populating of the repository*. Create editors to support the instantiation of the metamodels of artifacts. Furthermore, these tools include mechanisms to validate the conformity of the modeling artifact and publish the results into the repository using the appropriate interfaces.
- *Create tools to support the access to the repository*. Create views on the repository according to its APIs, its organization and the needs of the targeted system engineering process. For instance, a keyword-based search access tool is implemented for the *TeresaRepository*.
- *Create tools to support the administration of the repository*. Create editors to support the administration of the repository and the evolution of existing model libraries, users, artifacts and relationships.

4.3. Steps for deployment and configuration

We identified several roles. The modeling expert interacts with the repository to specify the modeling artifacts and then stores these artifacts, and the domain expert interacts with the repository to instantiate and then reuse these artifacts. The repository manager is responsible for the repository administration. Finally, the system developer selects the modeling artifact for building an application. Fig. 8 shows the process to be followed to prepare the usage of the repository.

Installation and deployment. Using the script developed during the implementation of the repository, the repository software system is deployed on an appropriate host, and the accompanying

development tools are installed in the user development environment. Assuming that all of the hardware and software resource requirements are met, the first step is the download and deployment of the repository server application. Once the server is started, we perform the repository initialization, comprised of the creation of the structure of the repository as well as the initialization and management of the compartments and users. Then, we proceed with the installation of the client tool suite.

Define access security. Once the repository server has been deployed, an access policy is added to specify resources, roles and access rules. For each user, access rights to compartments are added, which are filtered when such a user logs onto the repository system. Furthermore, the scope of access can be limited to a group of modeling artifacts.

Create models. Create instances of the modeling artifact metamodels and publish the results in the repository using the appropriate editors. During this activity, the pattern artifacts are built conforming to the pattern modeling language. An activity is added at this point to check the design conformity of the pattern.

Generate reports and documentation. At this point, the modeling artifact designer may generate documentation. If the pattern has been correctly defined, i.e., it conforms to the pattern modeling language, the pattern is ready for publication to the model-based repository. Otherwise, the issues from the report can be identified, and the pattern can be re-built by correcting or completing its relevant constructs. In addition, this task will be used during the usage of the repository to generate development environment-specific and multi-target documentation.

Define relationships between models. We provide a set of facilities for the repository organization allowing for the enhancement of its usage. For instance, we provide features to support the management of the relationships among artifact specifications and between artifact specifications and their complementary models. Each artifact is studied to identify its relationships with the other artifacts belonging to the same application domain with respect to the engineering process activity in which it is consumed. In our case, the goal of this activity is to organize patterns and give them a structure of a set of systems of patterns. For instance, a pattern is linked with other patterns and associated with property models using a predefined set of reference types.

Reuse of existing artifacts. Once the repository system is available, it serves an underlying trust engineering process through access tools, conforming to the process model shown in Fig. 11.

5. Architecture and implementation tools

In this section, we propose an MDE tool chain to support the proposed approach and assist the developers of a repository system, thus assisting the developers of software systems based on the repository. As discussed below, the proposed tool chain is designed to support the proposed metamodels; hence, the tool chain and the remainder of the activities involved in the approach may be developed in parallel. Appropriate tools for supporting our approach must satisfy the following key requirements:

- Enable the creation of the UML class diagrams used to describe the repository metamodel and the artifact metamodels.
- Support the implementation of a repository to store models and the related model libraries for classification and relationships.
- Enable the creation of models and the related model libraries and publication of the results into the repository.
- Support the administration and the internal management of the repository.
- Enable the creation of visualizations of the repository to facilitate its access.
- Enable the creation of application models.
- Enable transformations of the models from the repository format into the target-modeling environment.
- Enable the integration of application models and models imported from the repository.

To satisfy the above requirements, we define four integrated sets of software tools:

- *Toolset A* for populating the repository,
- *Toolset B* for retrieval and adaptation from the repository,
- *Toolset C* to serve as the repository software, including its administration and internal management, and
- *Toolset D* as the augmented target development environment.

This software system, including the specification, over target technology, evolution and maintenance for acquiring organizations, and end-users and front-end support provider, is detailed in the following. For illustration purposes, the described tool suite is related to the *TeresaRepository*. Specifically, we develop SEMCOMDT[5] (SEMCO model development tools) as an MDE tool chain to support all of the steps in our approach. We implemented the proposed approach as a set of Eclipse plug-ins as a proof of concept. 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.[6] In addition, a video tutorial presenting the tool suite is provided under the SEMCO Video Tutorial.

All of the presented metamodels, including the repository structure and interfaces as well as the S&D pattern metamodel and property metamodel, are specified using the EMF. The design tools are generated semi-automatically from these metamodels. Several enhancements are added to the generated code, such as creation wizards, to guide the modeling artifact designer in populating the repository. Visual enhancements are added to facilitate the recognition of different concepts as a first step toward a future visual syntax. The QVT operational language is used to describe the model transformations. The repository is implemented using the Eclipse CDO framework. SEMCOMDT offers the following features:

- *Gaya* for specifying and implementing a repository to store models,
- *Tiqueo* for specifying models of S&D properties conforming to *GPRM*,
- *Arabion* for specifying patterns that conform to *SEPM* (see Fig. 9),
- *Admin* for the repository management (see Fig. 10),
- *Retrieval* for the repository access (see Fig. 15).

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. The repository APIs are implemented as CDO clients and provided as an Eclipse plugin. The implementation is mainly based on the automatic code generation from the API model defined above. The generated Java code defines the different interfaces and functions provided by the repository APIs. The skeletons of the APIs implementations are then manually completed based on CDO technology. For more details, the reader is referred to [20].

For populating purposes, we build two design tools, (1) the property designer (Tiqueo), to be used by a *property designer*, and

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

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

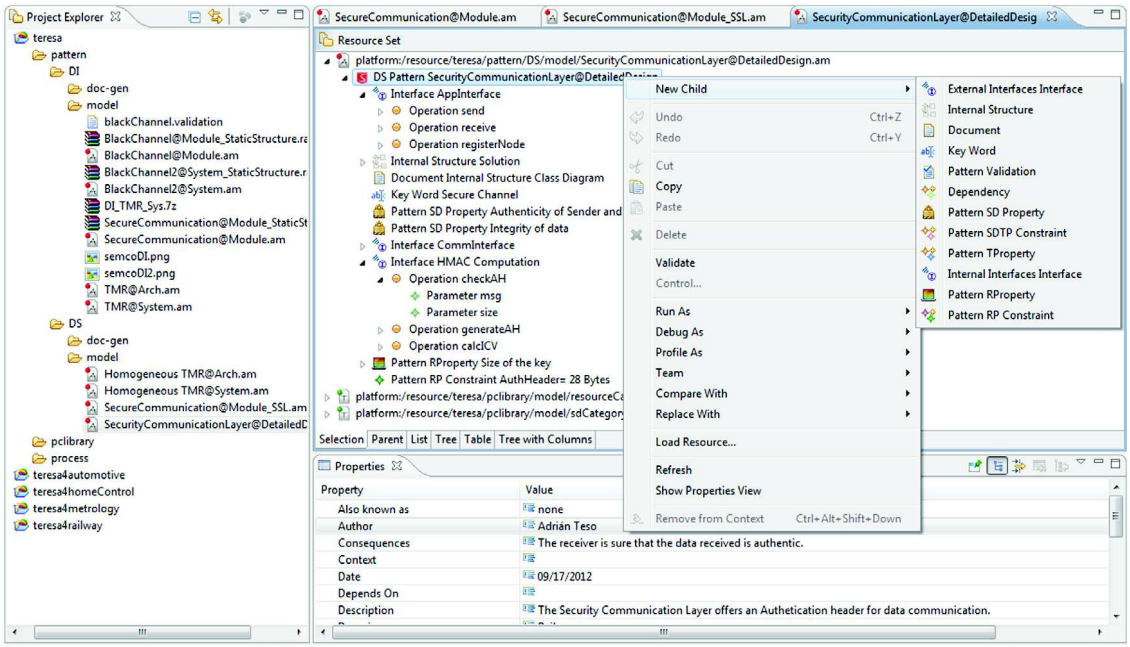

Fig. 9. Designing a pattern.

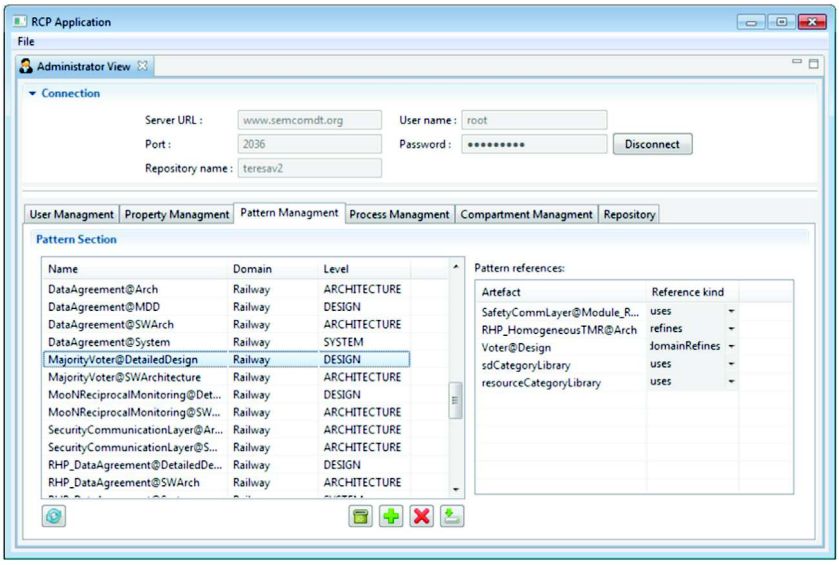

Fig. 10. Repository management and re-organization.

(2) the pattern designer (Arabion), to be used by a *pattern designer*. Tiqueo (resp. Arabion) interacts with the Gaya repository for publication purpose using the *Gaya4Property* (resp. *Gaya4Pattern*API). The user applications for populating the repository are implemented as a set of EMF tree-based editors to create patterns and the required libraries and are provided as Eclipse plugins.

To access the repository, which is to be used by a *system engineer*, the tool provides a set of facilities to help in selecting appropriate patterns, including *keyword* search, *lifecycle stage* search and property categories. The Tool includes features for exportation and instantiation as dialogues targeting domain-specific development environments. Moreover, the tool includes dependency-checking mechanisms. For example, a pattern cannot be instantiated when a property library is missing, and an error message will be thrown. We also provide software, as a Java-based GUI application, to support the management of the relationships among artifact specifications and between artifact specifications and their complementary models.

6. Framework for software system modeling artifacts

The proposed approach promotes model-based development coupled with a repository of modeling artifacts. This approach aims to define an engineering discipline to enforce reuse and to share expertise. The main goal of this section is to define a modeling framework to support the packaging of a set of modeling artifacts for system software engineering.

6.1. A modeling artifact-based development process

In our work, we promote a new discipline for system engineering around a model-based repository of modeling artifacts. The proposed framework addresses two types of processes: the process of *modeling artifacts development* and *system development with modeling artifacts*. The main concern of the first process is designing modeling artifacts for reuse, and the second concern is finding

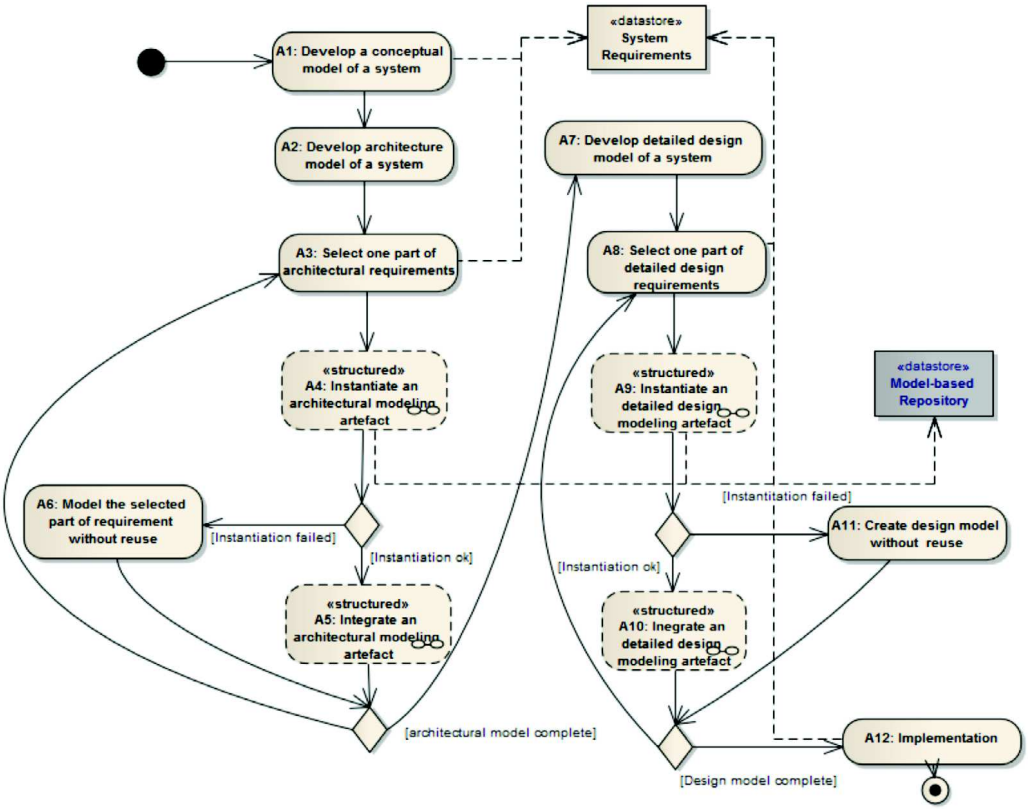

Fig. 11. The modeling artifact-based development process.

adequate modeling artifacts and evaluating them with regard to the requirements of the system under development. Therefore, we add a repository as a tier that acts as an intermediate agent between these two processes. A repository should provide a modeling container to support modeling artifact lifecycles associated with different methodologies.

Once the repository is available (the repository system is populated with modeling artifacts), it serves an underlying engineering process. In the process model visualized in Fig. 11 as an activity diagram, the developer starts with system specification (A1), fulfilling the requirements. In a traditional approach (non-repository-based approach) the developer would continue with the architecture design, module design, implementation and test. In our vision, instead of following these phases and defining new modeling artifacts, which typically consumes large amounts of time and effort and is prone to errors, the system developer merely needs to select appropriate modeling artifacts from the repository and integrate them in the system under development.

For each phase, the system developer executes the search/select from the repository to instantiate modeling artifacts in its modeling environment (A4 and A9) and integrate them in its models (A5 and A10) following an incremental process. The model specified in a certain activity $A(n-1)$ is then used in activity An. In the same manner, for a certain development stage n, the modeling artifacts identified previously in stage $(n-1)$ will help during the selection activity of the current phase. Moreover, the system developer can use a modeling artifact design process to develop their own solutions when the repository fails to deliver an appropriate modeling artifact at this stage. The software designer does not necessarily need to use one of the previously included artifacts stored in the repository. He can define custom software architecture for some modeling artifact (components) and avoid using the repository facilities (A6 and A11).

6.2. Application to a railway system case study

In this section, we report on an industrial case study performed in the railway domain. The case study enables us to determine that the model-based repository of patterns approach leads to a reduced number of or simplification of the engineering process steps and helps assess whether domain experts agree on the benefit of adopting the model-based repository approach in a real industrial context.

6.2.1. Overview of the case study

We demonstrate the applicability of our proposed framework through the *Safe4Rail* demonstrator, which is a simplified version of a real European Train Control System (ETCS) signaling, control and train protection system. The main functionality of this demonstrator is to supervise that the travel speed and distance do not exceed the authorized maximum values provided by the railway infrastructure. *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 importantly, the emergency brake must be activated when something goes wrong.

As shown in Fig. 12, the entire system is composed of:

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

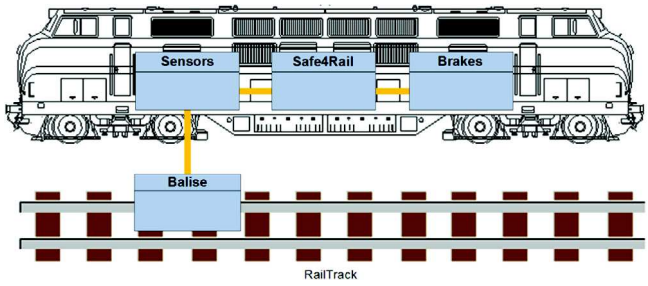

Fig. 12. Railway control system.

The fundamental functionality of the system is based on a set of inputs from the rail track, 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.

6.2.2. Description of the application and identification of patterns

There are three main use cases of the *Safe4Rail*system, which are described below. Fig. 13 provides a diagram of selected use cases, illustrating their relationships and their classification as safety-relevant or non-safety-relevant. We describe only a small part of this system, and we do not show the resulting model here, as it contains proprietary information from our industrial partner.

1. Activate the emergency brake and realize diagnostics (when the system is in Standby mode).
2. Supervise the train speed and position (when the system is in Supervision mode).
   (a) Estimate the current position and speed.
   (b) Supervise the current position and speed and activate warnings and brakes accordingly.
   (c) Provide information to the User.
3. Change between Standby and Supervision modes.

6.2.3. Repository of S&D patterns for the railway domain

The architects analyze system safety and security requirements and identify possible security and safety patterns to be used. Table 1 shows the list of patterns that are going to be used in the railway demonstrator, whereas Fig. 14 provides an overview of the railway S&D pattern system.

We used the Tiqueo editor and Arabion editor to create the corresponding property libraries and the set of patterns, respectively. Arabion uses the property libraries provided by Tiqueo to type the pattern property. Finally, we used the Gaya manager tool to set the relationships between the patterns. Fig. 14 provides an overview of the railway S&D pattern system.

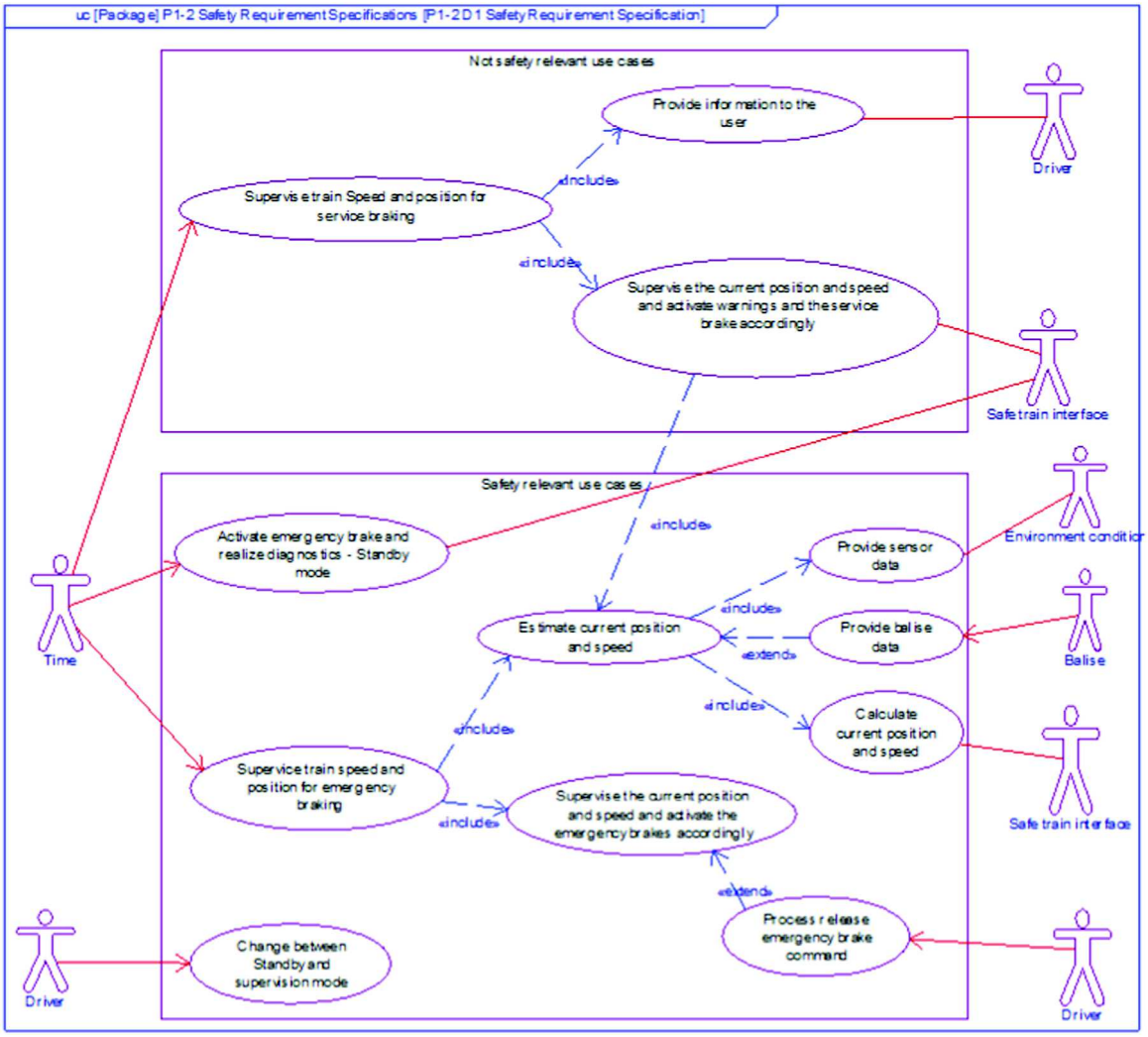

Fig. 13. Part of the Safe4Rail use-case diagram.

Table 1
List of patterns.

Pattern	Origin
TMR (Triple Modular Redundancy)	IEC-61508-2 (Table 2, 3), EN-50129 (Table E.4)
Majority voter	IEC-61508-2 (Table A.2, A.3 A.4)
Data agreement protocol	Book "Real-Time Systems:Design Principles for Distributed Embedded Applications"
Black channel	IEC-61508-2
Hypervisor	EN-50129 (Table E.4; Separation of safety-related systems from non safety-related systems)
Watchdog	EN-50126, IEC-61508-3 (Table A.11)
Reciprocal comparison	IEC-61508-2 (Table A.4), EN-51028 (Fault Detection & Diagnosis)
Safety communication layer	IEC-61784 / IEC-62280
Security communication layer	Book "Security patterns in practice: Building secure architectures using software patterns"

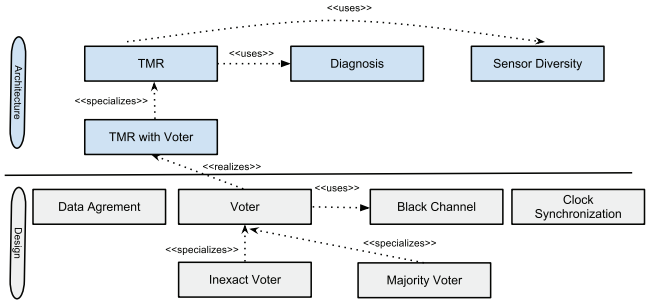

Fig. 14. Railway pattern system—overview.

6.2.4. Access tool for the railway domain

A customized access tool to facilitate the connection between the railway development environment and TERESA repository was developed. For this purpose, the access tool implements the *Gaya4SystemDeveloper* API provided as an Eclipse plugin and offers a GUI that allows the user to search and select patterns. When a pattern is selected, the access tool instantiates the pattern in the domain-specific tool. Because this task is carried out during product development, the instantiated pattern must be compliant with the current phase of the domain process and the user tools. By accessing the repository, we provide features based on model transformation techniques to adapt the model of the pattern to the target development environment. In the TERESA railway use case, the target format is a subset of UML, which can be imported by Rational Rhapsody.

The left part of Fig. 15 shows the main window for the railway access tool user interface.

This window has two panels: one for search and the other for display. The display panel on the bottom shows the name of the selected pattern and the different associated views, for instance, an example of a graphic view with a class diagram of the pattern implementation. The repository is searched by using either the "Name" field to enter part of the name or "Keywords" to enter some characteristics of the desired pattern. To instantiate the selected pattern, as shown in the right part of Fig. 15, the Access Tool will create a new representation of the selected pattern as a UML package for UML Rational Rhapsody using model transformation techniques.

6.2.5. System developer view point: Reuse of existing artifacts

In this process model, as shown in Fig. 16, the developer starts with requirement engineering/specification, followed by system specification. For each phase, the system developer executes the search/select from the repository to instantiate appropriate patterns in its modeling environment using the Access tool and then integrates them in its models following an incremental process. Moreover, the system developer can use the pattern designer tool (Arabion) to develop their own solutions when the repository fails to deliver the appropriate patterns in this phase.

The process flow for the example can be summarized with the following steps:

- Once the requirements are properly captured and imported into the development environment, such as Rhapsody, the repository may suggest possible patterns to meet general or specific S&D needs (according to the requirements and application domain): e.g., if the requirements contain the keywords *Redundancy* or *SIL4*, a suggestion could be to use a *TMR* pattern at the architecture level. In addition, some diagnosis techniques imposed by the railway standard may be suggested:
  - TMR (suggested by the tool),
  - Diagnosis techniques (suggested by the tool),
  - Sensor Diversity (searched by the System Architect).
- Based on the selected patterns, the repository may suggest related or complementary patterns. For instance, if the TMR has been integrated, the following patterns may be pro-posed in a second iteration, for instance, in the design phase:
  - Data Agreement
  - Voter
  - Black Channel
  - Clock Synchronization.

The system architecture shown in Fig. 17 specifies the Safe4Rail system decomposition, the relationship between the different blocks that compose the system. At this stage, the *system architect* makes some architectural decisions (based on the safety concept and requirements) and accesses the repository to search for and import suitable refined and specialized design patterns of interest:

1. The TMR is implemented with
   (a) Three homogeneous nodes ("*SupervisionNode*") connected to two Ethernet switches in a star topology, composed of:
      i. a computing unit microcontroller ("*ComputingUnit*", "*Microcontroller*"),
      ii. an "*FPGA*" that provides safety and non-safety related inputs/outputs ("IO Safety" and "*IO No Safety*"),
      iii. a *watchdog* as a pattern,
      iv. a software application ("*SupervisionApplication*") executed at the computing unit that integrates the *Safety Communication Layer* and *Secure Communication Layer* as patterns to support safe and secure communication among replicated communication channels,
      v. the usage of a *hypervisor* ("*DI_SA_BL_Hypervisor*") as a pattern to enable the integration in a single software application ("*SupervisionApplication*") of
         - safety software, such as software functionalities and safety techniques previously described in the safety concept,
         - non-safety-related application software such as the communication stack of the black channel.
   (b) Two Ethernet switches ("*EthernetSwitch*") associated with a black channel.
2. A single (safety) balise reader ("*BaliseReader*").
3. A black channel communication protocol (Ethernet/EtherCAT).

During this phase, new design patterns have been imported from the repository based on the system architect's decisions. For example, the selection of a supervision node with a single microcontroller has led to the use of a hypervisor to integrate safety- and non-safety-related software in a single microcontroller.

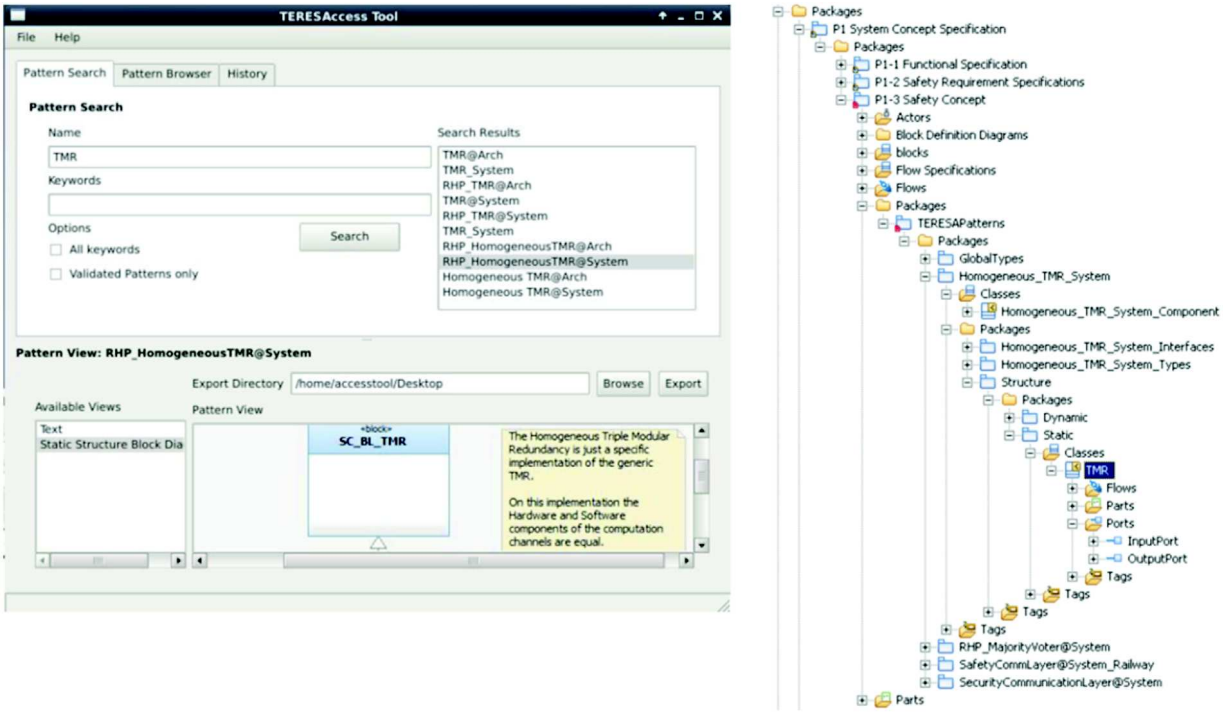

Fig. 15. Railway access tool.

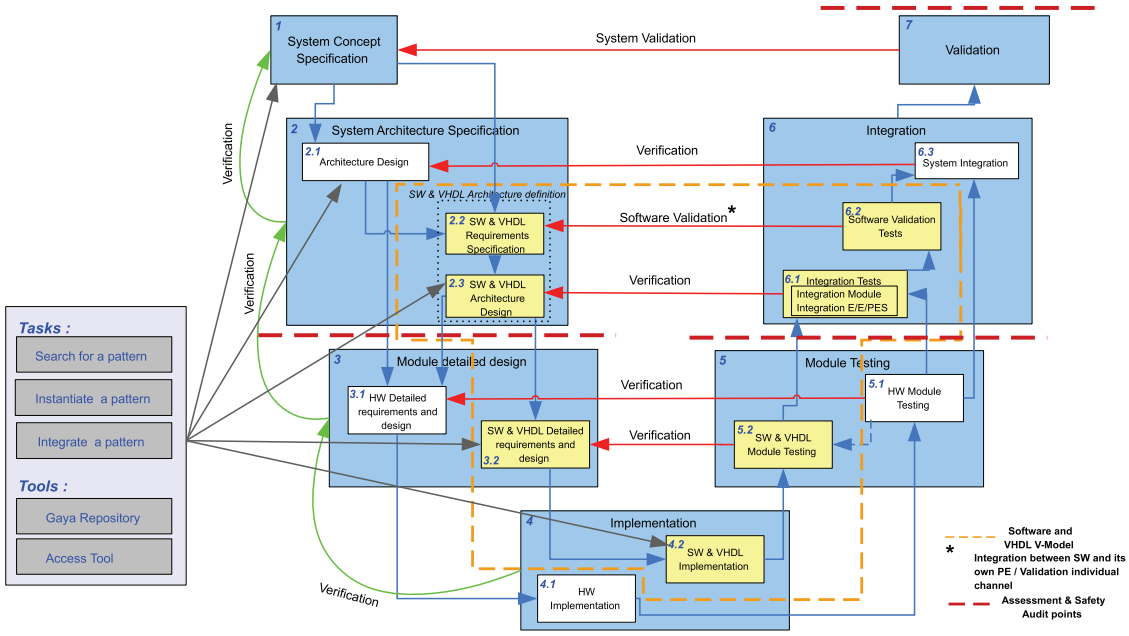

Fig. 16. Overview of the railway reference process model.

6.2.6. *Discussion*

In this subsection, the adaptation of railway processes to incorporate the model-based repository of patterns approach has been described. We test which of the provided tools are able to support the pattern integration and assist in the engineering process. In this context, the extendibility of the pattern repository for new patterns, as well as the extendibility of existing patterns, is also observed. Furthermore, we evaluate the degree to which the patterns are useful for increasing engineering productivity.

The S&D knowledge comprised in a pattern (e.g., in the form of guidelines and source code) will be observed with respect to its generality, i.e., we attempt to prove whether the same guidelines can be used to successfully instantiate the TERESA sector-specific engineering processes and whether they can also be used to instantiate other processes. We intend to demonstrate that the model-based repository of patterns approach leads to a reduced number of or to a simplification of the engineering process steps. The guidelines that are provided should support the developer regarding S&D issues and reduce the error frequency. We demonstrate that the application of the proposed approach theoretically yields important benefits for development engineers. This statement was proven in practice through the real implementation of the demonstrators.

The objective is to demonstrate the process flow and integration of the tools in the domain tool chains, not to solve the low-

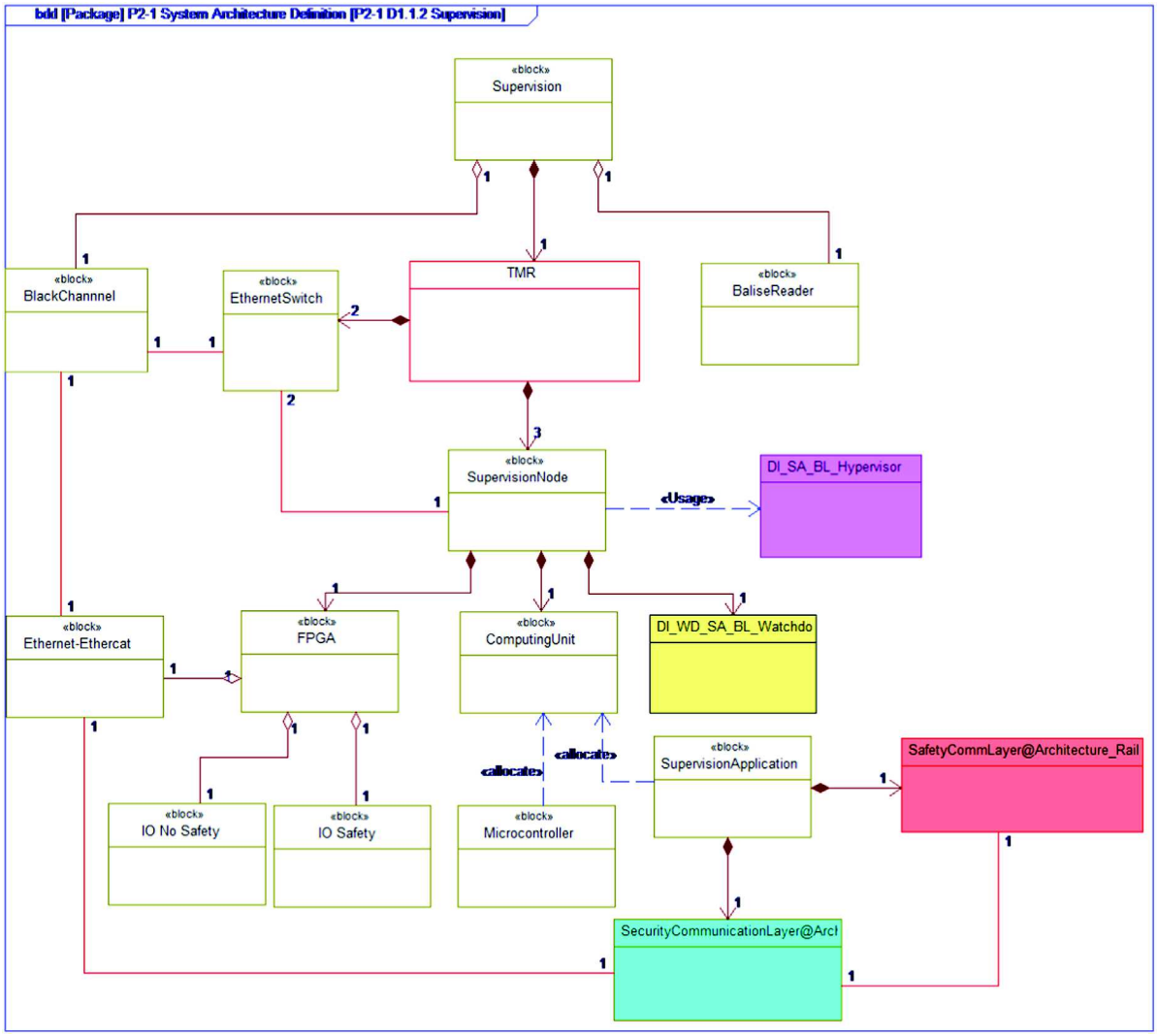

Fig. 17. Overview of the Safe4Rail system architecture.

level details of the approach integration. In this respect, there is considerable work to be done. For instance, in the railway domain, some examples of the modeling artifact (e.g., pattern) representation at different phases have been proposed, but these representations must be completely refined to allow for a straightforward integration of these modeling artifacts into the project models. The required modeling artifact representation at the same level may differ from one domain to another, so the Repository Access Tool is responsible for providing the information in the required format. The layout of the Repository Access Tool depends on the sector particularities, so a new "skin" must be defined every time a new sector is considered.

Projects for the development of embedded railway systems are commonly implemented by multidisciplinary teams that may be geographically dispersed. Therefore, the existence of a repository with pre-engineered patterns (designed and verified) can reduce the time-to-market (and overall project cost), facilitate complexity management, decrease the probability of systematic faults and improve the robustness of the developed products. These advantages are attained predominantly because each design pattern provides a concise representation of a concept (e.g., redundancy, as specified by IEC-61508) and/or well-known and pre-engineered solutions that naturally fit into the relevant engineering lifecycle.

7. Related work

In Model-Driven Development (MDD), model repositories [2,21,22] are used to facilitate the exchange of models through tools by managing modeling artifacts. Model repositories are often built as a layer on top of existing technologies (e.g., databases).

To lighten the queries to the repository, metadata can be added to select the appropriate artifacts. Therefore, there are some repositories that are composed solely of metadata. For instance, as presented in the standard ebXML [23] and an ebXML Repository Reference Implementation [24], a service repository can be seen as a metadata repository that contains metadata about location information to find a service. In [22], the authors proposed a reusable architecture decision model for setting-up model and metadata repositories. They aimed to design data model and metadata repositories. In addition, some helpers are included in the product to select a basic repository technology, choose appropriate repository metadata, and select suitable modeling levels of the model information stored in the repository. In [25], the authors proposed a repository implementation with support for storing and managing of artifacts. The supported artifacts are: metamodels, models, constraints, specifications, transformation rules, code, templates, configuration or documentation, and their metadata.

Moogle [26] is a model search engine that uses UML or a domain-specific language meta-model to create indexes that allow

Table 2
Solutions analysis.

Approaches	Representation flexibility	User-level interaction	Tool support	Methodology engineering oriented
Specific reuse repository [31,27,29]	***	$n+$	$n+$	****
Code dependences repository [32–36]	$n+$	**	$n+$	**
Metadata repository [23,24]	$n+$	$n+$	***	**
Model repository [2,30,21,22,26,37,28]	$n-$	$n+$	$n+$	$n-$
Our approach	$n+$	$n+$	$n+$	$n+$

for the evaluation of complex queries. Its key features include searching through different types of models (as long as their metamodel is provided). The index is built automatically, and the system attempts to present only the relevant parts of the results, for example, by trying to remove the XML tags or other unreadable characters to improve readability. The model elements' type and attributes and the hierarchy between model elements can be used as search criteria. Models are searched by using keywords (simple search), specifying the types of model elements to be returned (advanced search) and using filters organized into facets (browse). The user must know the metamodel elements to use the advanced search engines properly. Moogle uses the Apache SOLR ranking policy of the results. The most important information of the results is highlighted to make it clearer to the user.

The MORSE project [27] proposes a Model-Aware Service Environment repository to facilitate dynamic services for reflection models. MORSE addresses two common problems in MDD systems: traceability and collaboration. The model repository is the main component of MORSE and has been designed with the goal of abstracting from specific technologies. MORSE focuses on runtime services and processes and their integration and interaction with the repository.

The work described in [28] is a general-purpose approach that uses graph query processing to search repositories of models represented as graphs. First, the repository models are translated into directed graphs. Then, the system receives a query conforming to the considered DSL metamodel. To reduce the matching problem into a graph matching one, the submitted query is also transformed into a graph. Matches are calculated by finding a mapping between the query graph and project graphs or sub-graphs, depending on the granularity. The results are ranked using the graph edit distance metric via the A-Star algorithm. The prototype considers the case of the domain-specific WebML language.

The work in [29] presents a survey of business process model repositories and their related frameworks. This work addresses the management of large collections of business processes using repository structures and provides common repository functions, such as storage, search and version management. It targets the process model designer and allows for the reuse of process model artifacts. A comparison of process model repositories is presented to highlight the degree of reusability of artifacts.

Another issue is graphical modeling tool generation, as studied in the GraMMi project [30]. In this project, the repository is based on three levels of abstraction (metametamodel, meta-model and model). The repository stores both metamodels (notation definitions) and models (instantiation definitions). The repository access is enabled by an interface provided by itself.

GraMMi's Kernel allows for the management of persistent objects. Thus, this kernel aims to convert the objects (models) into an understandable form for the user via the graphical interface.

Compared to other work, our approach considers general metamodels and offers a generic transformation between the metamodel and implementation platforms. Furthermore, our approach does not focus on the implementation but rather on the methodology to develop the repository of models. Our implementation of the repository structure and interfaces is based on the EMF and a CDO-based repository. However, other existing technologies to support repository implementations may be used in our work as target platforms for repository generation. Our work aims to provide a model-based methodology for developing a repository of models that effectively supports reuse and interconnects the process of model specification and system development with models. The metamodel and methodology described in this paper may be used to specify the management and use of several types of repositories, as detailed above. In fact, model aspects or the overall assets of these repositories can be seen as artifacts supported by our metamodel. For instance, a process model aspect or the overall process model of the listed process models can be seen as artifacts supported by our metamodel. In return, the vision of the business process model repositories [29] may be used to manage the process element-type libraries.

Table 2 shows a comparison of the existing approaches and how difficult it is to implement/describe the most common concepts related to repositories in system and software engineering. The scale goes from natively supported ($n+$) and natively supported but not easy to use ($n-$), past usable but requires adaption (* * * *), down to nearly impossible to use (*).

8. Conclusion

Repositories of modeling artifacts have recently gained increasing attention for the enforcement of reuse in software engineering. In fact, repository-centric development processes are more adopted in software/system development, such as architecture- or pattern-centric development processes. The proposed framework for building a repository is based on metamodeling, which allows the structure of a repository and modeling artifacts to be specified at different levels of abstraction, and model transformation techniques for generation purposes. Moreover, we have proposed an operational architecture for the implementation of a repository. In addition, the tool suite promotes the separation of concerns during the development process by distinguishing the roles of the different stakeholders. Mainly, the access to the repository is customized regarding the development phases, the stakeholder's domain and his system knowledge.

The approach for developing a repository of models is generic; however, the discussion and implementation in this paper is based on the Eclipse platform, Ecore and a CDO-based repository. We have implemented a prototype named SEMCOMDT to support the approach as an Eclipse plug-in. Thus, this prototype plug-in supports the methodology defined in Section 4. The approach has been evaluated in the context of the TERESA project to build a repository of S&D patterns and property models to support the development of a railway signaling, control and train protection system.

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 simulations are important types of artifacts that we can consider in our framework to serve the systematic construction of large complex systems with multiple concerns. From another perspective, formal modeling language is used as a complementary approach when verification and validation are needed. For example, in security engineering, we may use formal languages, such as the Security Modeling

Framework (SeMF) [38,39], to represent the model of the solution and the security properties to enable verification. In the context of the TERESA project, [40] built the xSeMF language to provide a syntactically and semantically clear and consistent way to describe the security properties of systems or patterns. The metamodel behind xSeMF is specified using the EMF, and the xText[7] framework was used for the realization of a textual editor. Therefore, the xSeMF language comes with a syntax-aware editor for the Eclipse platform for manipulation. This ensures compatibility with the remainder of the repository and pattern tooling, as it utilizes the same modeling technology platform.

This similarity facilitates the integration in the Tool chain. It also avoids media discontinuity and the need for an additional tool. As a result, specification languages (semi-formal and formal), roles and compartments related to each of them can be clearly defined and applied in system development for increased flexibility and efficiency. The appropriate tools will also be adapted accordingly.

In future work, we plan to study the automation of the search and instantiation of models, for which a framework for simpler constraint specifications would be beneficial. The expected goal is to highlight the content of the repository in the form of a map representing modeling artifact dependences. In addition, we will study the integration of our tooling with other MDE tools. Additionally, we will seek new opportunities to apply the framework to other domains. Other issues are that

- 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 compartments and not around the artifact.
- There are CDO-based implementation limitations.
- The effort of creating the repository and artifacts must be evaluated.

Acknowledgments

This work was initiated in the context of the SEMCO project. It is supported by the European FP7 TERESA project and by the French FUI 7 SIRSEC project. The authors are particularly grateful to Adel Ziani and Jacob Geisel for their valuable help in the implementation and development of SEMCO tools. In addition, the authors would like to thank TERESA consortium members for their participation in the implementation of the use case.

References

[1] S. Ravi, A. Raghunathan, P. Kocher, S. Hattangady, Security in embedded systems: Design challenges, ACM Trans. Embed. Comput. Syst. 3 (3) (2004) 461–491.
[2] P.A. Bernstein, U. Dayal, An overview of repository technology, in: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB'94, Morgan Kaufmann Publishers Inc., 1994, pp. 705–713.
[3] V. Burégio, E. de Almeida, D. Ludrédio, S. Meira, A reuse repository system: From specification to deployment, in: H. Mei (Ed.), High Confidence Software Reuse in Large Systems, in: Lecture Notes in Computer Science, vol. 5030, Springer, Berlin, Heidelberg, 2008, pp. 88–99.
[4] TERESA, TERESA Project, Trusted Computing Engineering for Resource Constrained Embedded Systems Applications, http://www.teresa-project.org/.
[5] D. Schmidt, Model-driven engineering, IEEE comput. 39 (2) (2006) 41–47.
[6] I. Crnkovic, M.R.V. Chaudron, S. Larsson, Component-based development process and component lifecycle, in: Proceedings of the International Conference on Software Engineering Advances, ICSEA 2006, IEEE Computer Society, 2006, p. 44.
[7] W. Frakes, K. Kang, Software reuse research: Status and future, IEEE Trans. Softw. Eng. 31 (7) (2005) 529–536.
[8] B. Hamid, SEMCO Project, System and software Engineering for embedded systems applications with Multi- COncerns support, http://www.semcomdt.org.
[9] J. Gray, J.-P. Tolvanen, S. Kelly, A. Gokhale, S. Neema, J. Sprinkle, Domain-Specific Modeling, Chapman & Hall/CRC, 2007.
[10] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework 2.0, second ed., Addison-Wesley Professional, 2009.
[11] OMG, Meta Object Facility (MOF) 2. 0 Query / View / Transformation Specification, 2008.
[12] B. Hamid, A model-driven methodology approach for developing a repository of models, in: Model and Data Engineering—4th International Conference—MEDI 2014, in: Lecture Notes in Computer Science, vol. 8748, Springer, 2014, pp. 29–44.
[13] B. Hamid, N. Desnos, C. Grepet, 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.
[14] C. Alexander, S. Ishikawa, M. Silverstein, A Pattern Language, in: Center for Environmental Structure Series, vol. 2, Oxford University Press, New York, NY, 1977.
[15] A. Ziani, B. Hamid, S. Trujillo, Towards a unified meta-model for resources-constrained embedded systems, in: 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), IEEE, 2011, pp. 485–492.
[16] B. Hamid, S. Gurgens, C. Jouvray, N. Desnos, Enforcing S&D pattern design in RCES with modeling and formal approaches, in: J. Whittle (Ed.), ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS, Vol. 6981, Springer, 2011, pp. 319–333.
[17] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Dependable Secure Comput. 1 (2004) 11–33.
[18] D. Riehle, H. Züllighoven, Understanding and using patterns in software development, TAPOS 2 (1) (1996) 3–13.
[19] G. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture: A System of Patterns, Vol. 1, John Wiley and Sons, 1996.
[20] B. Hamid, A. Ziani, J. Geisel, Towards tool support for pattern-based secure and dependable systems development, in: ACadeMics Tooling with Eclipse, ACME, Montpellier, France, ACM DL, 2013, pp. 1–6.
[21] R.B. France, J.M. Bieman, B.H.C. Cheng, Repository for model driven development, ReMoDD, in: MoDELS Workshops'06, 2006, pp. 311–317.
[22] C. Mayr, U. Zdun, S. Dustdar, Reusable architectural decision model for model and metadata repositories, in: FMCO, 2008, pp. 1–20.
[23] ebXML: Oasis Registry Services Specification v2.5, 2003.
[24] freebXML: Oasis ebxml registry reference implementation project, 2007. http://ebxmlrr.sourceforge.net/.
[25] N. Milanovic, R.-D. Kutsche, T. Baum, M. Cartsburg, H. Elmasgünes, M. Pohl, J. Widiker, Model&Metamodel, Metadata and Document Repository for Software and Data Integration, in: MoDELS, 2008, pp. 416–430.
[26] D. Lucrédio, F. de Mattos, P. Renata, J. Whittle, MOOGLE: A model search engine, in: MoDELS, Springer, 2008, pp. 296–310.
[27] T. Holmes, U. Zdun, S. Dustdar, MORSE: A model-aware service environment, 2009.
[28] B. Bislimovska, A. Bozzon, M. Brambilla, P. Fraternali, Graph-based search over web application model repositories, in: ICWE, Springer, 2011, pp. 90–104.
[29] Z. Yan, R.M. Dijkman, P. Grefen, Business process model repositories—framework and survey, Inf. Softw. Technol. 54 (4) (2012) 380–395.
[30] C. Sapia, M. Blaschka, 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, IEEE Computer Society, 2000, p. 8058.
[31] H.B. Hadji, K. Su-Kyoung, C. Ho-Jin, A Representation Model for Reusable Assets to Support User Context, in: Service-Oriented System Engineering, 2008. SOSE'08. IEEE International Symposium on, 2008, pp. 91–96.
[32] Apache Software Foundation, Ivy, 2015. http://ant.apache.org/ivy/.
[33] Apache Software Foundation, Maven, 2015. https://maven.apache.org/what-is-maven.html.
[34] GRADLE INC., Gradle, 2015. https://gradle.org/why/robust-dependency-management/.
[35] Bundler Core Team, Bundler, 2015. http://bundler.io/.
[36] D.L. Berre, P. Rapicault, Dependency management for the eclipse ecosystem: eclipse p2, metadata and resolution, in: Proceedings of the 1st International Workshop on Open Component Ecosystems, ACM, 2009, pp. 21–30.
[37] C. Hein, T. Ritter, M. Wagner, Model-driven tool integration with modelbus, in: Workshop Future Trends of Model- Driven Development, 2009, pp. 50–52.
[38] S. Gürgens, P. Ochsenschläger, C. Rudolph, On a formal framework for security properties, Int. Comput. Stand. Interface J. (CSI) 27 (5) (2005) 457–466. Special issue on formal methods, techniques and tools for secure and reliable applications.
[39] B. Hamid, S. Gürgens, A. Fuchs, Security patterns modeling and formalization for pattern-based development of secure software systems, Innov. Syst. Softw. Eng. 11 (3) (2015) 1–32. http://dx.doi.org/10.1007/s11334-015-0259-1.
[40] J. Eichler, A. Fuchs, N. Lincke, Supporting security engineering at design time with adequate tooling, in: 2012 IEEE 15th International Conference on Computational Science and Engineering, CSE, 2012, pp. 194–201.

[7] https://eclipse.org/Xtext/.

Brahim Hamid is an associate professor at the University of Toulouse Jean-Jaurés and he is a member of the IRIT-MACAO team. He got his Ph.D. degree in 2007 in the area of dependability in distributed computing systems from the University of Bordeaux (France). In addition, he has an M.Sc. in Theoretical Computer Science that provides him with background on mathematical, logic and formal concepts. He has been an assistant professor (ATER) at ENSEIRB (Bordeaux, France), and a member of LaBRI (France). Then he worked as a post-doc in the modeling group at the CEA-Saclay List (France). He was a visiting professor at the university of Concordia (August 2011), at the university of Florida (September 2014) and at the university of Vienna (April 2015). His main research topics are software languages engineering, at both the foundations and application level, particularly for resource constrained systems. He works on security, dependability, software architectures, formalization, validation and verification as well as supporting reconfiguration. Furthermore, he is an expert in model-driven development approaches both in research and teaching. Emphasis of his work lies on the development of tools to model and to analyze secure and dependable software architecture of critical infrastructures as railway and metrology systems. He has participated in a number of national and European research projects. In particular, he has led successfully the IRIT effort on the TERESA FP7 European project, and several national projects. Brahim Hamid is author or co-author of over 50 internationally reviewed publications, mostly on software engineering and IT security and dependability, and he has co-organized several international workshops (DANCE, SD4RCES). He serves as a reviewer in numerous leading journals of the software engineering domain (SOSYM, ADHOC networks, JSA, JSS, JSME, SPE, DIST, etc.), and as a member of various international conference program committees (SERE, EUROPLOP, SEAA, SERENE, etc.). He has been invited to do expertise work for various organizations: FWF, FIT (Austria) and for various national research programs (ANR, CIR, etc.). He is also participating in several working groups and involved in several teaching activities related to security and system engineering, as well as engaging technology transfer to other organizations and other bodies or agencies, and more generally through consulting and training activities.