A Model-Driven Methodology Approach for Developing a Repository of Models

Brahim Hamid

A Model-Driven Methodology Approach for Developing a Repository of Models

Brahim Hamid

IRIT, University of Toulouse
118 Route de Narbonne, 31062 Toulouse Cedex 9, France
hamid@irit.fr

Abstract. 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. The wanted role of models is to ease, systematize and standardize the approach of the construction of software-based systems. In order to enforce reuse and to interconnect the process of models' specification and the system development with models, we promote a model-based approach coupled with a repository of models. In this paper, we propose a Model-Driven Engineering methodological approach for the development of a repository of models and an operational architecture for development tools. In particular, we show the feasibility of our own approach by reporting some preliminary prototype providing a model-based repository of security and dependability (S&D) pattern models.

Keywords: Modeling artifact, Repository, Meta-model, Model Driven Engineering, Embedded Systems, Pattern.

1 Introduction

It is widely acknowledged that designers and developers of new-generation embedded systems are facing an exponential effort to manage the continuous increasing requirements of such systems [22]. 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. The integration of these features requires the availability of both application domain specific knowledge and feature expertise at the same time. 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.

Model Driven Engineering (MDE) based solutions seem very promising to meet the needs of trusted embedded system applications development. The idea promoted by MDE 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. MDE allows to increase software quality and to reduce the software systems development life cycle. Moreover, from a model, it is possible to

automatize some steps by model refinements and generate code for all or parts of the application. Domain Specific Modeling Languages (DSML) [8] has recently increased in popularity to cover a wider spectrum of concerns. As we will see, such a process reuses many practices from Model Driven Engineering. For instance, metamodeling and transformation techniques.

We believe that the use of a repository providing constructs for componentization of modeling artifacts can provide an efficient way to address these problems, improving the industrial efficiency and fostering technology *reuse* across domains (reuse of models at different levels), reducing the amount of effort and time needed to design a complex system. According to Bernstein and Dayal [3], a repository is a shared database of information on engineered artifacts. They introduce the fact that a repository has (1) a *Manager* for modeling, retrieving, and managing the components 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: 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 embedded systems. Here, we describe a methodological approach for the creation of a flexible repository of modeling artifacts and for managing the models in that repository. To show the feasibility of our approach, we are developing an operational implementation in the context of the FP7 TERESA project [6]. Besides in this task some services dedicated to repository features will be developed. The goal is to integrate features together thanks to model-based repository engineering coupled with MDE technology; hence this will attempt to leverage reuse of model building blocks from the repository.

The rest of this paper is organized as follows. In Section 2, we discuss the modeling framework around a repository of modeling artifacts. Section 3 presents modeling language to support the design of the repository structure and its interfaces. In Section 4, we describe the approach for designing 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 the usage of the defined modeling framework in the context of 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 draws future work directions.

2 The 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. 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 processes levels.*

Adapting the definition of pattern language given by Christopher Alexander [1], 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 objective.*

In our work, we promote a new discipline for system engineering around a model-based repository of modeling artifacts. The proposed framework addresses two kind 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 one is finding the adequate modeling artifacts and evaluating them with regard the system-under-development's requirements. Therefore, we add a repository as a tier which acts as intermediate agent between these tow processes. A repository should provide a modeling container to support modeling artifacts life-cycle associated with different methodologies.

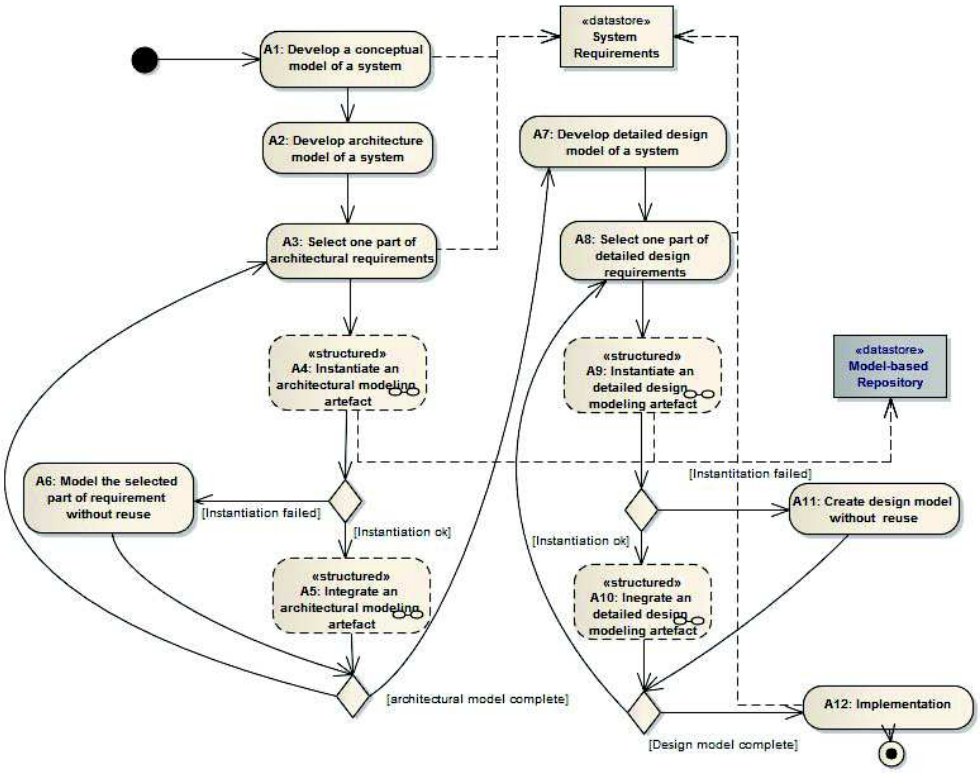

Fig. 1. The Modeling Artifact-based Development Process

Once the repository is available (the repository system populated with modeling artifacts), it serves an underlying engineering process. In the process model visualized in Fig. 1, as activity diagram, the developer starts by 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, that usually are time and efforts consuming, as well as errors prone, 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 integrates 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 way, for a certain development stage n, the modeling artifacts identified previously in stage (phase) $n-1$ will help during the selection activity of a current phase. Moreover, the system developer can use a modeling artifact design process to develop their own solutions when the repository fails to deliver appropriate modeling artifact at this stage. It is important to remark that the software designer does not necessarily need to use one of the artifacts stored in the repository previously included. He can define custom software architecture for some modeling artifact (components), and avoid using the repository facilities (A6 and A11).

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 in order to use them in engineering processes (see Fig. 7).

3.1 System and Software Artifact Repository Model Specification(SARM)

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 modeling artifacts and to manage 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 in details the meaning of principal concepts used to specify the repository:

- SarmRepository. Is the core element used to define a repository.
- SarmCompartment. Is used for the categorization of the stored data. We have identified two main kinds of compartments:
  - SarmSpecLangCompartment. Is used to store the specification languages (SeSpecLang) of the modeling artifacts (SeArtefact).
  - SarmArtefactCompartment. Is used to store the modeling artifacts. To simplify the identification of a modeling artifact regarding the software development stage in which it's involved, an SeArtefact has an lifecycleStage typed with an external model library SeLifecycleStage.

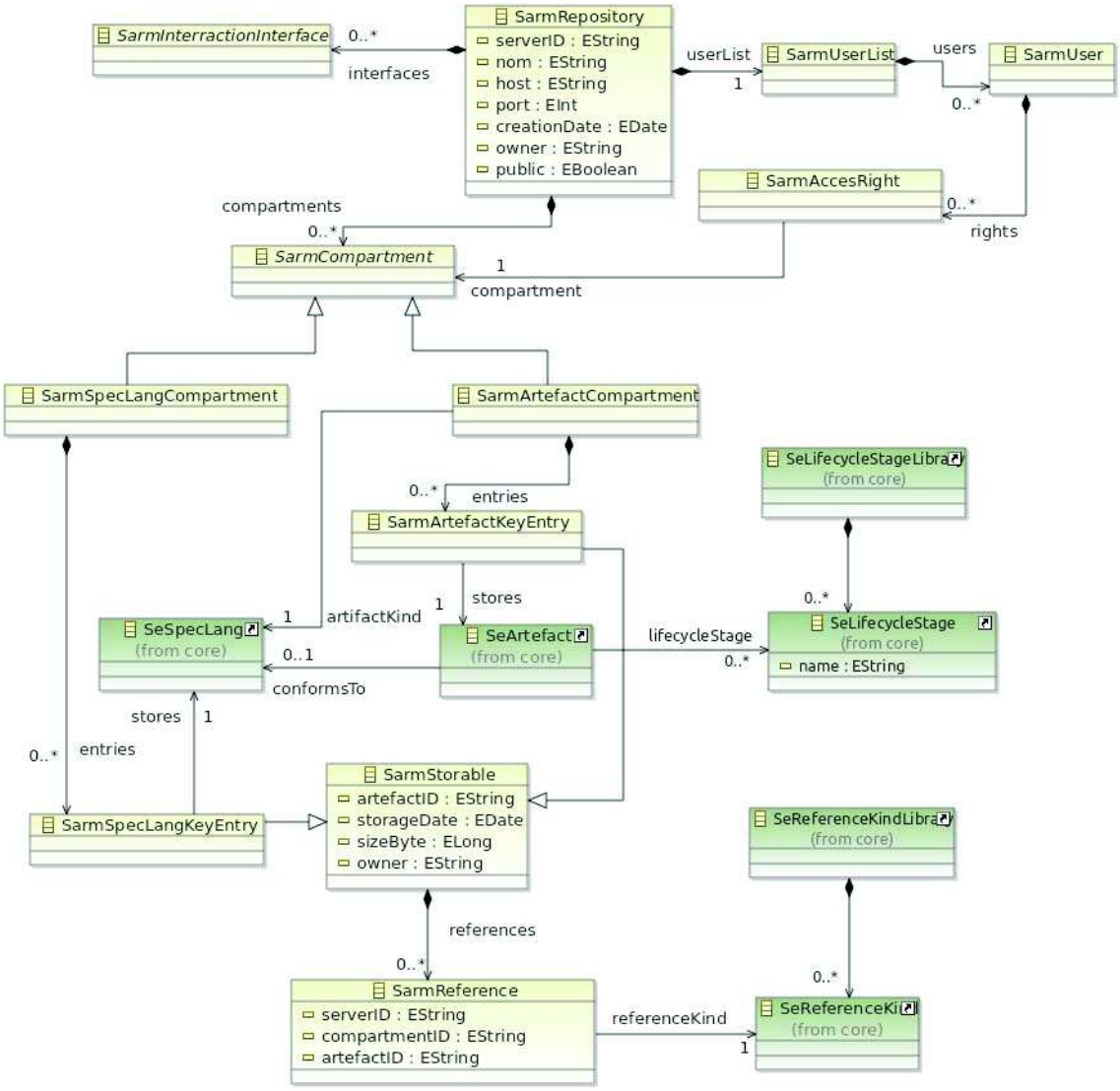

Fig. 2. The Repository Specification Metamodel (SARM)

- SarmStorable. Is used to define a set of characteristics of the model-based repository content, mainly those related to storage. We can define: *artefactURI*, *storageDate*, *sizeByte*, etc. . .
  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 and external model library SeReferenceKind.
- *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.
- *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.

For the interaction purposes, the repository exposes its content through a set of interfaces (SarmInteractionInterface), as depicted 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 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.

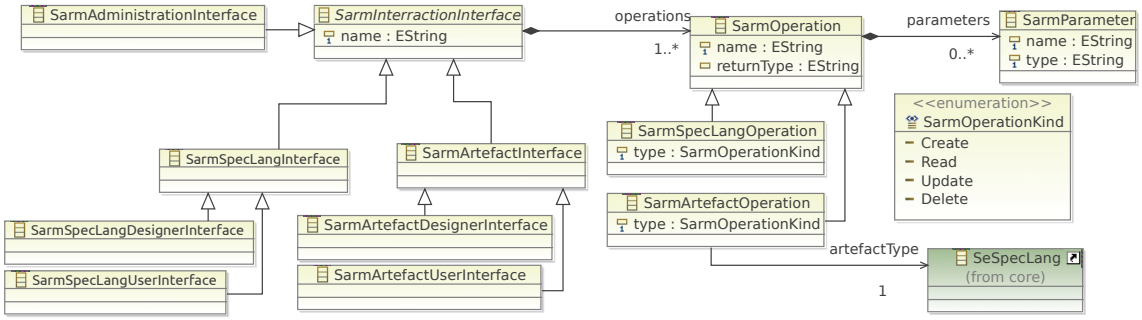

Fig. 3. The Repository API Specification Metamodel

4 Methodology

In this section, we describe a methodological approach for the creation of a flexible repository of modeling artifacts and for managing the models in that repository, such as visualized in Fig.4.

For illustration purpose, we will focus in the rest of the paper on the repository of security and dependability patterns, which acts as a specific demonstration for the TERESA resource constrained embedded systems, called *TeresaRepository*.

The following sections introduce the example of the *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 whole cycle from the creation of the artifacts' metamodels, the instantiation of the repository metamodel, the instantiation of these metamodels as modeling artifact for populating the repository, the management of the repository, and an overview on how the resulting repository software will support system engineering process.

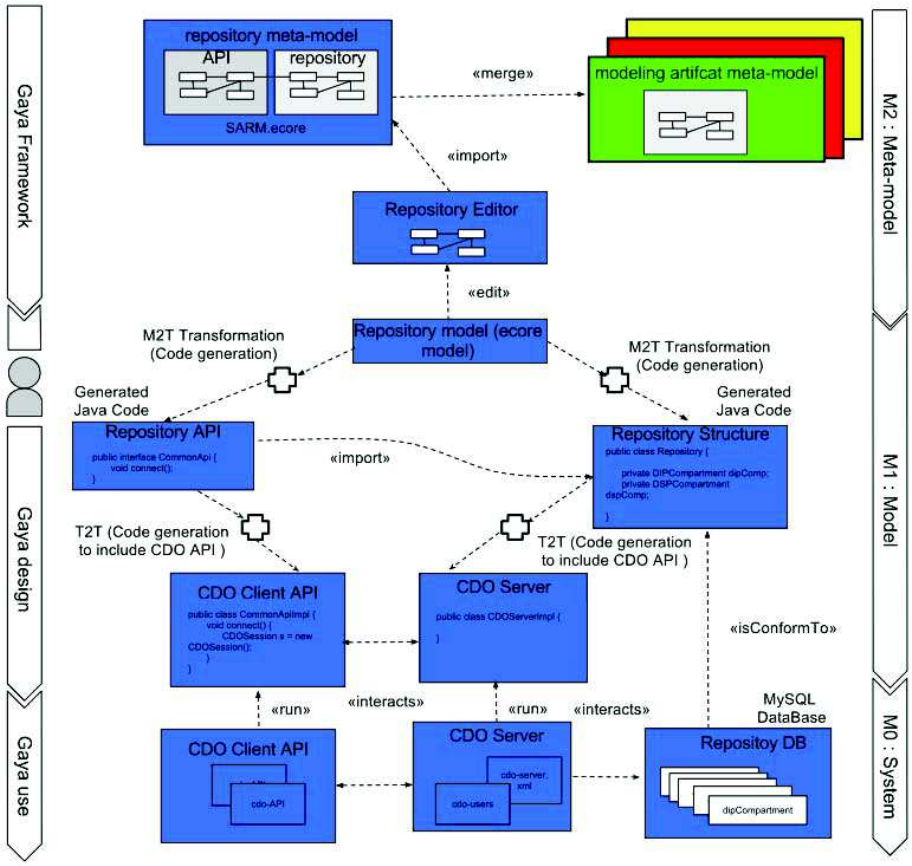

Fig. 4. Overview of the model-based repository building process

4.1 An S&D Pattern Repository

In the context of the TERESA project, we deal with three kinds of modeling artifacts: S&D patterns, S&D property models and resource property models. In this vision, the S&D pattern, derived from (resp. associated with) domain specific models, aims at helping the system engineer to integrate application S&D building blocks. Now, we briefly describe the modeling languages used to specify these artifacts. For more details, the reader is referred to [21] and [9] for property modeling language and for pattern modeling language, respectively.

4.1.1 Generic Property Modeling Language (GPRM)

The Generic PRoperty Metamodel (GPRM) [21], which is depicted with the Ecore notations in Fig. 5, is a metamodel defining a new formalism (i.e. language) for describing property libraries including units, types and property categories. For instance, security and dependability attributes [2] such as authenticity, confidentiality and availability are

defined as categories. These categories require a set of measures types (degree, metrics, ...) and units (boolean, float,...). For that, we instantiate the appropriate type library and its corresponding unit library. These models are used as external model libraries to type the properties of the patterns. Especially during the editing of the pattern we define the properties and the constraints using these libraries.

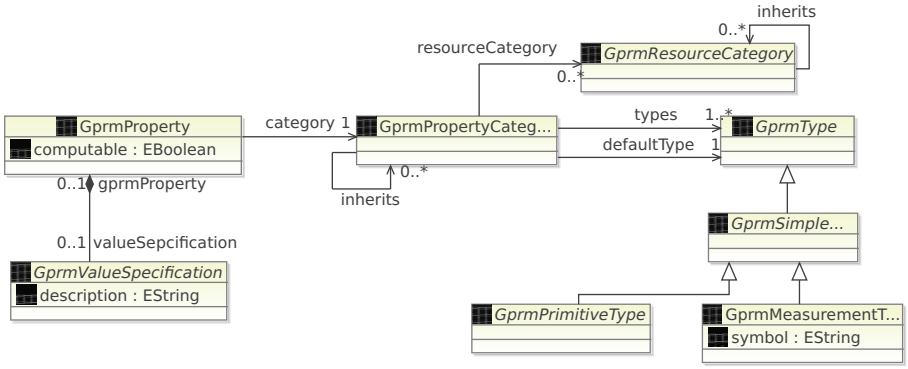

Fig. 5. The (simplified) GPRM Metamodel

4.1.2 Pattern Specification Metamodel

The System and Software Engineering Pattern Metamodel (SEPM) [9] 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 way to capture meta-information related to a pattern and its context of use. The following figure describes the principal concepts of the SEPM metamodel with the Ecore notations.

4.2 Model-Based Repository Building Process

(a) 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, SEPM and GPRM metamodels are created and stored as Ecore models (Fig. 5 and Fig. 6).

(b) Create tools to support the repository modeling process: Write editors for the specification of the repository structure and APIs.

(c) Specify model libraries for classifications of artifacts: At each stage of the system engineering development process, identify the appropriate modeling artifacts to use by classifying them. In our context, we use the pattern classification of Riehle and Buschmann [17,5], which is (1) *System Patterns*, *Architectural Patterns*, *Design Patterns* and *Implementation Patterns* to create the model library SeLifecycleStage.

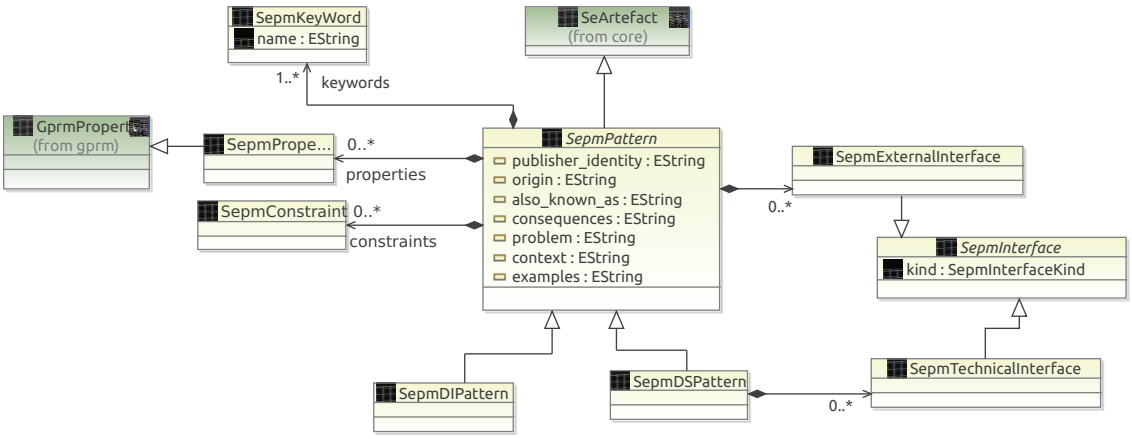

Fig. 6. The (simplified) SEPM Metamodel

(d) Specify model libraries for relationships between artifacts: At each stage n of the system engineering development process, the modeling artifacts identified previously in stage (phase) $n-1$ will help during the selection activity of a current phase. For instance, a pattern may be linked with other patterns and associated with property models using a predefined set of reference kinds. For example *refines*, *specializes*, *uses* etc. Here, we create the SeReferenceKind model library to support specifying relationships across artifacts.

(e) Specify the repository structure: Use the editors, the metamodels and the model libraries to instantiate the SARM metamodel to create the model of the repository comprising the creation of metamodels' compartments, the artifacts' compartments, the users' list etc. The structure of the repository and its APIs are then available to modelers for populating and managing 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 the property modeling language. We also define a set of compartments to store the artifacts. In addition to the repository structure, we present define the model of interfaces (APIs) to exhibit the content of the repository and its management.

(f) Create tools for generating code: The resulting model is then used as input for the model transformations in order to generate the repository and APIs software implementation targeting a specific technological platform, for instance CDO [1] (as shown in the middle part of Fig. 4). Also, specify scripts to perform the installation and deployment of the resulted repository software system.

[1] `http://www.eclipse.org/cdo/`

(g) Specify views on the repository for access tools: Creating views on the repository according to its APIs, its organization and on the needs of the targeted system engineering process. For instance, a key word based-search access tool is implemented for the *TeresaRepository*.

(h) Create tools to support the populating of the repository: Creating editors to support the instantiation of the metamodels of artifacts. Furthermore, these tools includes mechanisms to validate the conformity of the modeling artifact and to publish the results into the repository using the appropriate interface.

(i) Create tools to support the administration of the repository: Creating editors to support the administration of the repository, the evolution of existing model libraries, users, artifacts relationships etc.

4.3 Exploitation of the Repository

We identified several roles. The modeling expert interacts with the repository to specify the modeling artifacts and then to store these artifacts, and the domain expert interacts with the repository in order to instantiate and then to reuse these artifacts. The repository manager is responsible for the repository administration. Finally, the system developer selects the modeling artifact for building an application. The following steps depicts the process to be followed to use the repository.

(a) Installation and deployment: The repository software system is deployed on an appropriate host while the accompanying development tools are installed in the user development environment.

(b) Define access security: Create users' list and grant them access rights to compartments.

(c) Create models: Create instances of the modeling artifacts metamodels and publish the results into the repository using appropriate editors. During this activity the pattern artifacts were built conforming to the pattern modeling language. An activity is added at this point to check the design conformity of the pattern.

(d) Generate reports and documentation: At this point, the modeling artifact designer may generate documentation. If the pattern has been correctly defined, i.e conforms the pattern modeling language, the pattern is ready for the publication to the model-based repository. Otherwise, we can find the issues from the report and re-build the pattern by correcting or completing its relevant constructs.

(e) Define relationships between models: Create instances of artifacts relationships model libraries. Also, each artifact is studied in order 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. The goal of this activity, in our case, is to organize patterns, to give them a structure of a set of pattern systems.

(f) 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 visualized in Fig. 1.

5 Architecture and Implementation Tools

In this section, we propose an Model-Driven Engineering toc repository system, and hence to assist the developers of software provide four integrated sets of software tools: (i) *Tool set A* for pc (ii) *Tool set B* for retrieval from the repository, (iii) *Tool set C* as and (iv) *Tool set D* for managing the repository. The followir system from the specification, over target technology, evolutic acquiring organizations, end-users and front-end support provic

5.1 Tool-suite Architecture

To build our repository system, we use the well known archite architectures as an alternative client-server organizations, as sh

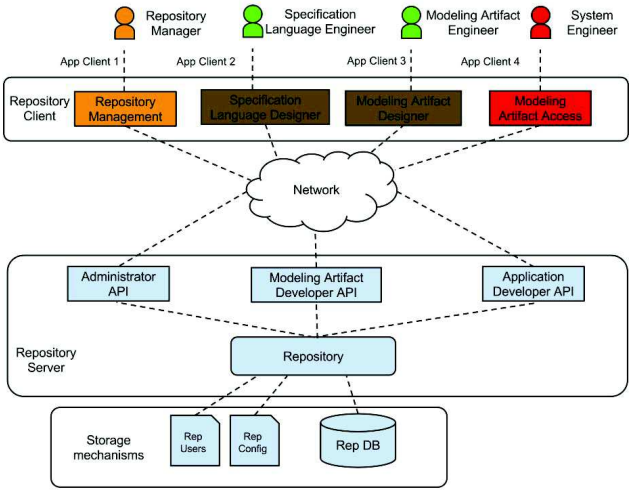

Fig. 7. An overview of repository system architecture

The server part (middle part of Fig. 7) is responsible for managing and storing the data interacting with storage mechanisms (down part of Fig. 7). In addition, the server part provides the implementation of the common APIs to interact with the repository content. For this, we identify a set of interfaces (APIs) for applications in order to create modeling artifact, in order to use them and in order to manage the repository. The user application part (top part of Fig. 7) is responsible for populating the repository and for using its content using the common APIs.

5.2 Implementation Details

Using the proposed metamodels, the Connected Data Objects (CDO) and the Eclipse Modeling Framework (EMF) [19], ongoing experimental work is done with SEMCOMDT [2] (SEMCO Model Development Tools, IRIT's editor and platform plugins). For our example, the tool-suite is composed of:

[2] `http://www.semcomdt.org`

– *Gaya.* for the repository platform (structure and interfaces) conforming to *SARM*,
– *Tiqueo.* for specifying models of S&D properties conforming to *GPRM*,
– *Arabion.* for specifying patterns conforming to *SEPM*,
– *Admin.* for the repository management,
– *Retrieval.* for the repository access.

The server part of *Gaya* is composed of two components: (1) *GayaServer* providing the implementation of the common API and (2) *GayaMARS* providing the storage mechanisms. The client part of *Gaya* provides interfaces, such as *Gaya4Pattern* (implements the API4PatternDesigner), *Gaya4Property* (implements the API4PropDesigner), *Gaya4Admin* (implements API4Admin) and *Gaya4SystemDeveloper* (implements the API4PatternUser). For pupulating purpose, we build two design tools, (1) The property designer (Tiqueo), to be used by a *property designer* and (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).

For accessing the repository, to be used by a *system engineer*, the tool provides a set of facilities to help selecting appropriate patterns including *key word* search, *lifecycle stage* search and property categories. 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.

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 based firstly on the automatic code generation from the APIs model defined above. The generated Java code defines the different interfaces and functions provided by the repository APIs. The skeleton of the APIs implementations are then completed manually based on CDO technology.

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 provided as Eclipse plugins. We also provide software, as a Java based GUI application, to manage the repository and for accessing. For more details, the reader is referred to [10].

6 Application of a Model-Based Repository of S&D Patterns to a Railway System Case Study

In the context of the TERESA project[3], we evaluated the approach to build an engineering discipline for trust that is adapted to RCES combining MDE and a model-based repository of S&D patterns and their related property models. 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

[3] `http://www.teresa-project.org/`

the patterns property. Finally, we used the Gaya manager tool to set the relationships between the patterns. The TERESA repository contains so far (on January 2014):

- *Compartments.* 21 compartments to store artifacts of the TERESA domains.
- *Users.* 10 users.
- *Property Libraries.* 69 property model libraries.
- *Pattern Libraries.* 59 patterns.

One of the case studies acting as TERESA demonstrators is set in the railway domain throw the Safe4Rail demonstrator. 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 process flow for the example can be summarized with the following steps:

- Once the requirements are properly captured and imported into the development environment, for instance *Rhapsody*, the repository may suggest possible patterns to meet general or specific S&D needs (according to 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 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 proposed in a second iteration, for instance at design phase:
  - Data Agreement
  - Voter
  - Black Channel
  - Clock Synchronization

7 Related Work

In Model-Driven Development (MDD), model repositories [13,7,3] 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 (for instance, databases).

In order to ease the query on the repository, metadata can be added to select the appropriate artifacts. Therefore, there exist some repositories that are composed solely of metadata. For instance, as presented in the standard ebXML [15] and an ebXML Repository Reference Implementation [16], a service repository can be seen as a metadata repository that contains metadata about location information to find a service. In [13], 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 for selecting a basic repository technology, choosing appropriate repository metadata, and selecting suitable modeling

levels of the model information stored in the repository. In [14], they proposed a repository implementation with storing and managing of artifacts support. The supported artifacts are: metamodels, models, constraints, specifications, transformation rules, code, templates, configuration or documentation, and their metadata.

Moogle [12] is a model search engine that uses UML or Domain Specific Language meta-model in order to create indexes that allow the evaluation of complex queries. Its key features include searching through different kind of models, as long as their metamodel is provided. The index is built automatically and the system tries to present only the relevant part of the results, for example trying to remove the XML tags or other unreadable characters to improve readability. The model elements type, attributes and hierarchy between model elements can be used as a search criteria. Models are searched by using keywords (Simple Search), by specifying the types of model elements to be returned (Advanced Search) and by using filters organized into facets (Browse). In order to properly use the advanced search engines, the user needs to know the metamodel elements. Moogle uses the Apache SOLR ranking policy of the results. The most important information of the results are highlighted to make them more clear to the user.

The MORSE project [11] proposes a Model-Aware Service Environment repository, for facilitating dynamically services to 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 to abstract from specific technologies. MORSE focuses on runtime services and processes and their integration and interaction with the repository.

The work described in [4] is a general-purpose approach using graph query processing for searching repository 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. In order to reduce the matching problem into a graph matching one, the submitted query is also transformed in a graph. Matches are calculated by finding a mapping between the query graph and the project graphs or sub-graphs, depending on the granularity. The results are ranked using the graph edit distance metric by means of the A-Star algorithm. The prototype considers the case of the domain-specific WebML language.

The work in [20] 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 artefacts.

Another issue is graphical modeling tool generation as studied in the GraMMi project [18]. In this project the repository is based on three levels of abstraction (metametamodel, metamodel and model). The repository stores both metamodels (notation definitions) and models (instantiation definitions). The repository access is made thanks to an interface provided by itself. GraMMi's Kernel allows to manage persistent objects. So this kernel aims at converting the objects (models) in an understandable form for the user via the graphical interface.

The metamodel and the methodology described in this paper may be used to specify the management and the use of these kinds of repositories. In fact, models aspects or the assets as a whole of the aforementioned repositories can be seen as artifacts supported by our metamodel. In return, the provided technologies to support repository implementations may be used in our work as targets platforms for repository generation.

8 Conclusion

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

The proposed framework for building a repository is based on metamodeling, which allows to specify the structure of a repository and modeling artifacts at different levels of abstraction, and model transformation techniques for generation purposes. Moreover, we 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 presented here has been evaluated in in the context of the TERESA project for a repository of S&D patterns and property models, where 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.

In a wider scope, new specification languages may be designed and stored with their related artifact in the repository. For instance, components, resources, analysis and simulations 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.

As future work, we plan to study the automation of the search and instantiation of models and a framework for simpler specification of constraints would be beneficial. In addition, we will study the integration of our tooling with other MDE tools. Also, we will seek new opportunities to apply the framework to other domains.

References

1. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language. Center for Environmental Structure Series, vol. 2. Oxford University Press, New York (1977)
2. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing 1, 11–33 (2004)
3. Bernstein, P.A., Dayal, U.: An Overview of Repository Technology. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB 1994, pp. 705–713. Morgan Kaufmann Publishers Inc. (1994)

4. Bislimovska, B., Bozzon, A., Brambilla, M., Fraternali, P.: Graph-based search over web application model repositories. In: Auer, S., Díaz, O., Papadopoulos, G.A. (eds.) ICWE 2011. LNCS, vol. 6757, pp. 90–104. Springer, Heidelberg (2011)
5. Buschmann, G., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: a system of patterns, vol. 1. John Wiley and Sons (1996)
6. TERESA Consortium. TERESA Project (Trusted Computing Engineering for Resource Constrained Embedded Systems Applications), `http://www.teresa-project.org/`
7. France, R.B., Bieman, J., Cheng, B.H.C.: Repository for Model Driven Development (ReMoDD). In: Kühne, T. (ed.) MoDELS 2006. LNCS, vol. 4364, pp. 311–317. Springer, Heidelberg (2007)
8. Gray, J., Tolvanen, J.-P., Kelly, S., Gokhale, A., Neema, S., Sprinkle, J.: Domain-Specific Modeling. Chapman & Hall/CRC (2007)
9. Hamid, B., Gürgens, S., Jouvray, C., Desnos, N.: Enforcing S&D Pattern Design in RCES with Modeling and Formal Approaches. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 319–333. Springer, Heidelberg (2011)
10. Hamid, B., Ziani, A., Geisel, J.: Towards Tool Support for Pattern-Based Secure and Dependable Systems Development. In: ACadeMics Tooling with Eclipse (ACME), Montpellier, France, pp. 1–6. ACM DL (2013)
11. Holmes, T., Zdun, U., Dustdar, S.: MORSE: A Model-Aware Service Environment (2009)
12. Lucrédio, D., de M. Fortes, R.P., Whittle, J.: MOOGLE: A model search engine. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 296–310. Springer, Heidelberg (2008)
13. Mayr, C., Zdun, U., Dustdar, S.: Reusable Architectural Decision Model for Model and Metadata Repositories. In: de Boer, F.S., Bonsangue, M.M., Madelaine, E. (eds.) FMCO 2008. LNCS, vol. 5751, pp. 1–20. Springer, Heidelberg (2009)
14. Milanovic, N., Kutsche, R.-D., Baum, T., Cartsburg, M., Elmasgünes, H., Pohl, M., Widiker, J.: Model&Metamodel, Metadata and Document Repository for Software and Data Integration. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 416–430. Springer, Heidelberg (2008)
15. Oasis. ebXML: Oasis Registry Services Specification v2.5 (2003)
16. Oasis. freebXML: Oasis ebxml registry reference implementation project (2007), `http://ebxmlrr.sourceforge.net/`
17. Riehle, D., Züllighoven, H.: Understanding and using patterns in software development. TAPOS 2(1), 3–13 (1996)
18. Sapia, C., Blaschka, M., Höfling, G.: 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, HICSS 2000, p. 8058. IEEE Computer Society (2000)
19. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0, 2nd edn. Addison-Wesley Professional (2009)
20. Yan, Z., Dijkman, R.M., Grefen, P.: Business process model repositories - framework and survey. Information & Software Technology 54(4), 380–395 (2012)
21. Ziani, A., Hamid, B., Trujillo, S.: Towards a Unified Meta-model for Resources-Constrained Embedded Systems. In: 37th EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 485–492. IEEE (2011)
22. Zurawski, R.: Embedded systems in industrial applications - challenges and trends. In: SIES (2007)