Model-Driven Engineering for Trusted Embedded Systems Based on Security and Dependability Patterns

Brahim Hamid[1], Jacob Geisel[1], Adel Ziani[1], Jean-Michel Bruel[1], and Jon Perez[2]

[1] IRIT, University of Toulouse
118 Route de Narbonne, 31062 Toulouse Cedex 9, France
{hamid,geisel,ziani,bruel}@irit.fr
[2] Ikerlan, Mandragon, Spain
jmperez@ikerlan.es

Abstract. Nowadays, many practitioners express their worries about current software engineering practices. New recommendations should be considered to ground software engineering on two pillars: solid theory and proven principles. We took the second pillar towards software engineering for embedded system applications, focusing on the problem of integrating Security and Dependability (S&D) by design to foster reuse. The framework and the methodology we propose associate the model-driven paradigm and a model-based repository of S&D patterns to support the design of trusted Resource Constrained Embedded System (RCES) applications for multiple domains (e.g., railway, metrology, automotive). The approach has been successfully evaluated by the TERESA project external reviewers as well as internally by the Ikerlan Research Center for the railway domain.

Keywords: Resource Constrained Embedded Systems, Security, Dependability, Repository, Pattern, Metamodel, Model Driven Engineering.

1 Introduction

The software of embedded systems [1] is not conventional software that can be built using usual paradigms. In particular, the development of resource constrained embedded systems (RCES) addresses constraints regarding memory, computational processing power and/or limited energy. Non-functional requirements such as Security and Dependability (S&D) [2] become more important as well as more difficult to achieve. The integration of S&D features requires the availability of both application domain specific knowledge and S&D expertise at the same time. In fact, capturing and providing this expertise by the way of S&D *patterns* can support embedded systems development.

In our previous work [3], we studied pattern modeling frameworks [4,5] and we proposed methods to model security and dependability aspects in patterns

F. Khendek et al. (Eds.): SDL 2013, LNCS 7916, pp. 72–90, 2013.

and to validate whether these still hold in RCES (Resource Constrained Embedded Systems) after pattern application. The question remains at which stage of the development process to integrate S&D patterns. In our work, we promote a new discipline for system engineering using a pattern as its first class citizen: Pattern-based System Engineering (PBSE). PBSE addresses challenges similar to those studied in software engineering. Closely related to our vision is the Component Based Software Engineering (CBSE) [6]. Therefore, PBSE focuses on patterns and from this viewpoint addresses two kind of processes: the process of *pattern development* and *system development with patterns*. The main concern of the first process is designing patterns for reuse and the second one is finding the adequate patterns and evaluating them with regard the system-under-development's requirements.

In this paper, we propose a methodology based on Model-Driven Engineering (MDE) and a model-based repository of S&D patterns for security and dependability engineering. At the core of the methodology is a set of Domain Specific Modeling Languages (DSML) [7] that allow modeling S&D patterns and repository structure. Such an approach, relying on an MDE too -suite supporting the methodology and thus in our context supporting automated model-based repository building and access in industry[1]. We discuss the benefits, such as reduced modeling effort and improved readability, achieved when applying the methodology to an industrial case study where we have used the modeling language to model the repository of S&D patterns for the domain of railway applications.

The rest of this paper is organized as follows. In Sect. 2, we present a review of the most important related work. In Sect. 3, we present the proposed methodology. Section 4 details the specification languages proposed in the context of the methodology. In Sect. 5, we introduce the tool-chain supporting the methodology. Then, in Sect. 6, we illustrate the methodology through the example of a railway application. Section 7 describes a first feedback on the methodology we propose. Finally, Sect. 8 concludes the paper with an outlook on future work.

2 Related Work

In developing software applications with security and dependability support, the use of patterns should lead to well structured applications. In [8] a hybrid set of patterns is used in the development of fault-tolerant software applications. These patterns are based on classical fault tolerant strategies such as N-Version programming and recovery block, consensus, voting. Extending this framework, [9] proposed a framework for the development of dependable software systems based on a pattern approach. They reused proven fault tolerance techniques in the form of fault tolerance patterns. The pattern specification consists of a service-based architectural design and deployment restrictions in form of UML deployment diagrams for the different architectural services.

In [10], a group of seven patterns is presented as a security framework for building applications. [11] introduced the concept of a security pattern system

[1] The approach is evaluated in the context of the TERESA project (`http://www.teresa-project.org/`).

as a set of patterns with a linkage between them and described how security patterns contribute to the security engineering process. [12] presented an extension of UML, called UMLsec, that enables to express security relevant information within diagrams in a system specification. UMLsec is defined in form of a UML profile using UML extension mechanisms, allowing the specification of security patterns and the integration of these patterns into system development.

Regarding the analysis aspects, [13] used the concept of security problem frames as analysis patterns for security problems and associated solution approaches. These frames are also grouped in a pattern system with a list of their dependencies. The analysis activities using these patterns are described with a highlight on how the solution may be set with a focus on the privacy requirement anonymity. For the software architecture, [14] presented an evaluation of security patterns in the context of secure software architectures. The evaluation is based on the existing methods for secure software development, such as guidelines, and on threat categories.

Another important issue is the identification of security patterns. [15] proposed a new specification template inspired on secure system development needs. The template is augmented with UML notations for the solution and with formal artifacts for the requirements properties.

In addition to the above, S&D patterns are studied as precise specifications of validated S&D mechanisms. [5] explains how this can be achieved by using a library of precisely described and formally verified security and dependability (S&D) solutions as mechanisms, while [16] reports an empirical experience, about the adopting and eliciting of these S&D patterns in the Air Traffic Management (ATM) domain. The results are of interest, mainly the use of patterns as a guidance to structure the analysis of operational aspects when they are used at the design stage. Recently [17] presented an overview and new directions on how security patterns are used in the whole aspects of software systems from domain analysis to the infrastructures.

3 Pattern-Based Security Engineering Methodology

We now present an overview of our modeling building processes as activity diagrams. In this description, we will give the main keys to understand why our process is based on a generic, incremental and a constructive approach. Additional and detailed information will be provided during the implementation of the related design environment. Moreover, we provide a set of definitions and concepts that might prove useful in understanding our approach. Then, we detail the description of the integrated process used for the development of the Safe4Rail application in Sect. 6.

3.1 Definitions

Adapting the definition of pattern language given by Christopher Alexander [18], we define the following:

Definition 1 (Modeling Artifact Language.). *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.*

Definition 2 (Security Pattern.). *A security pattern describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution [11].*

Definition 3 (Security Pattern Language.). *We define a security pattern language as a modeling artifact language where its constituent parts are security patterns and their relationships.*

Definition 4 (Instantiation.). *An instantiation activity takes a pattern and its related artifacts from the repository and adds it to the end-developer environment. This task enables the pattern to be used while modeling.*

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

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

Definition 5 (Integration.). *An integration activity happens within the development environment when a pattern and its related artifacts are introduced into an application design. Some development environments may come with native support for the integration.*

3.2 Development of Reusable Artifacts

The pattern development process supports a number of features including pattern design, validation, interaction with a verification framework, deposit to and retrieval and from the repository.

The process root, as shown in Fig. 1, indicates the start of the creation of a pattern (A1). It contains some initialization actions to define the pattern attributes (e.g, name, author, date,...). The next activity is the modeling of the pattern artifacts (A2) collecting data interacting with (1) *Domain knowledge and expertise* providing an informal pattern description and (2) the model-based Repository to refer to existing patterns. 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 (A3). The next activity (A4) deals with the pattern validation. It supports the formal validation of a pattern using an external process [3]. The result is a set of validation artifacts. At this point, the pattern designer may generate documentation (A6). If the pattern has been

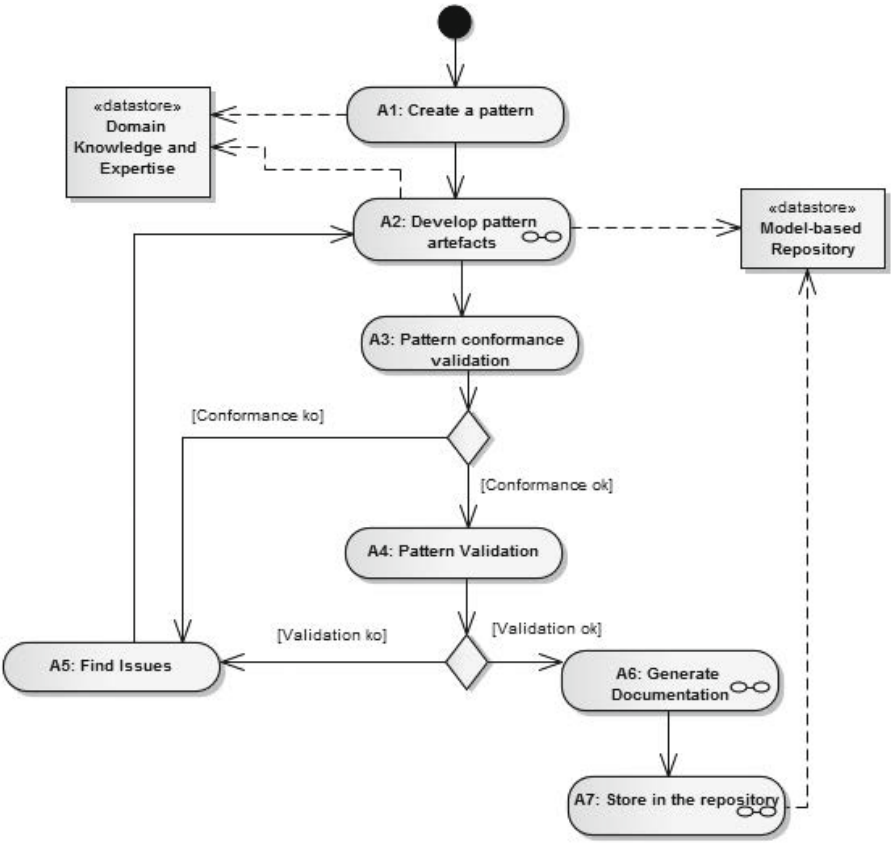

Fig. 1. Pattern development process

correctly defined (i.e. conforms to the pattern modeling language and is formally validated) the pattern is ready for the publication to the model-based repository (A7). Otherwise, we can find the issues and re-build the pattern (A5) by correcting or completing its relevant constructs.

3.3 Repository Designer View Point

The goal of this process is to organize the repository content, in our case patterns, to give them a structure of a set of pattern languages for application domains [19]. As visualized in the top part of Fig. 2, each pattern from a certain application domain is studied in order to identify its relationships with the other patterns belonging to the same application domain with respect to the engineering process' activity in which it is consumed (see the bottom part of Fig. 2).

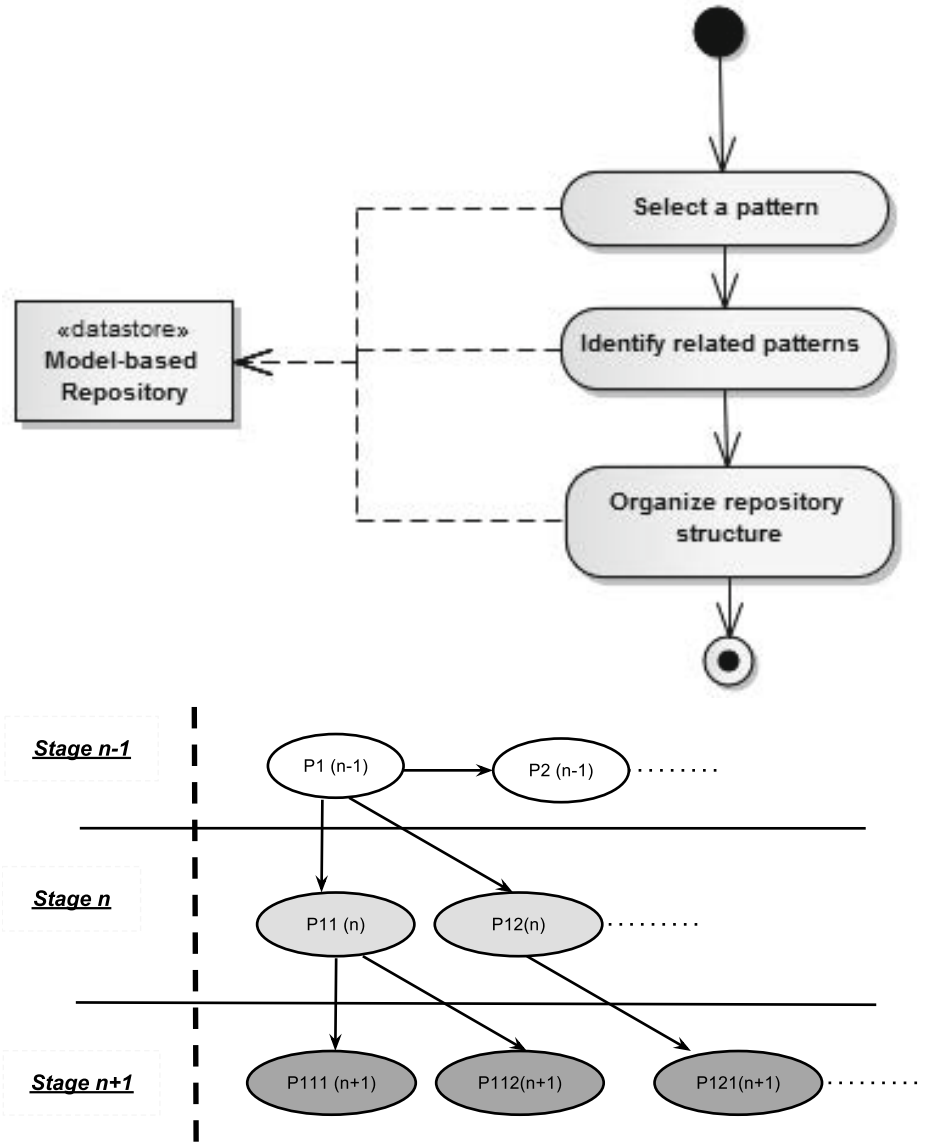

Fig. 2. Repository Process

3.4 Reuse of Existing Artifacts

Once the repository[2] is available, it serves an underlying trust engineering process. In the process model visualized in Fig. 3, the developer starts by system

[2] The repository system populated with S&D Patterns.

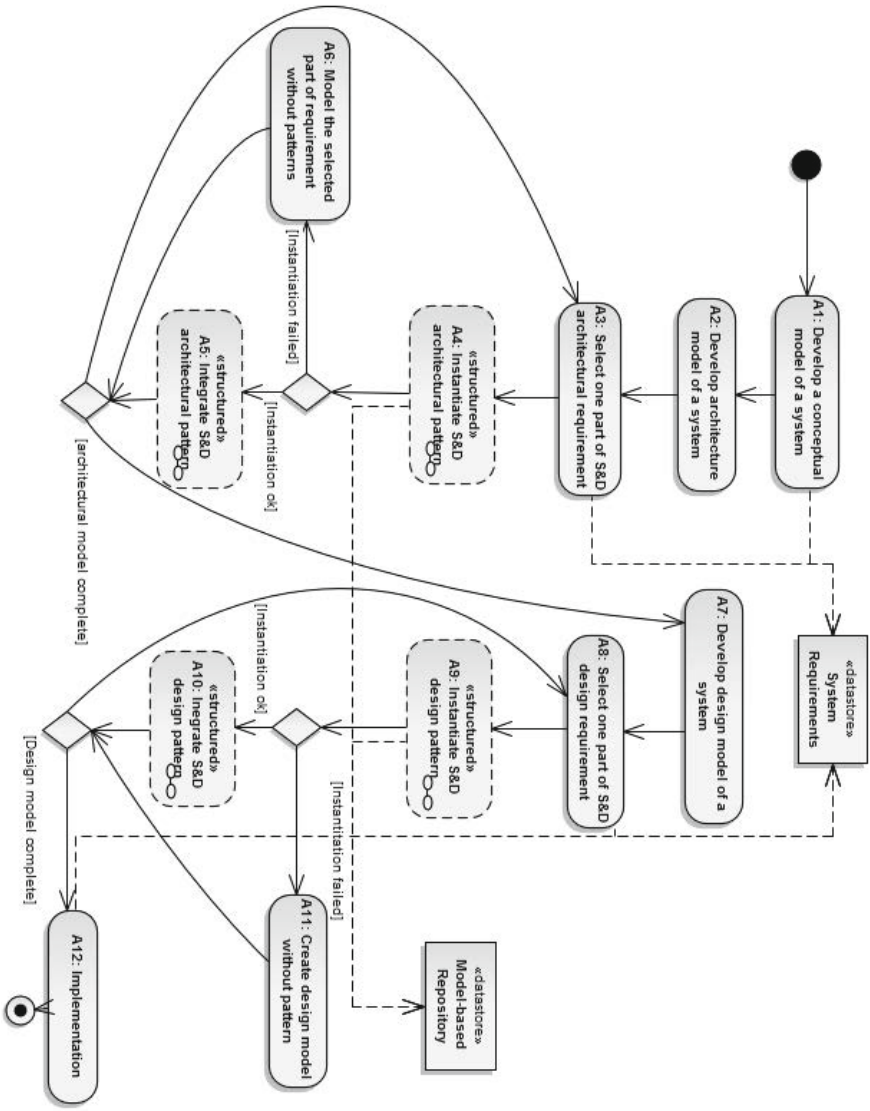

Fig. 3. The S&D Pattern-based Development Process

specification (A1) fulfilling the requirements. In a traditional approach (non pattern-based approach) the developer would continue with the architecture design, module design, implementation and test. In our vision, instead of following this phase and defining new modeling artifacts, that usually are time and effort consuming, as well as error prone, the system developer merely needs to select appropriate patterns 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 patterns 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 patterns identified previously in stage (phase) $n-1$ will help during the selection activity of a current phase. Moreover, the system developer can use the pattern design process, introduced previously, to develop their own solutions when the repository fails to deliver appropriate patterns 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 patterns (components), and avoid using the repository facilities (A6 and A11).

4 Specification Languages (DSLs)

In this section we present the specification languages to support the PBSE methodology: repository structure specification language (SARM) and pattern modeling language (SEPM).

4.1 Repository Structure Specification Language

A repository is a data structure that stores artifacts and that allows the user to publish and to select them for reuse and to share expertise. The specification of the structure of the repository is based on the organization of its content and the way it interacts with other engineering processes. The analysis of these requirements allows us to identify two main parts: the first one is dedicated to store and manage data in the form of *Compartments*, the second one is about the *Interfaces* in order to publish and to retrieve patterns and models.

The principal classes of the System and software Artifact Repository Metamodel (SARM) are described with Ecore notations in Fig. 4. The following part depicts more detailed the meaning of the principal concepts used to structure the repository:

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

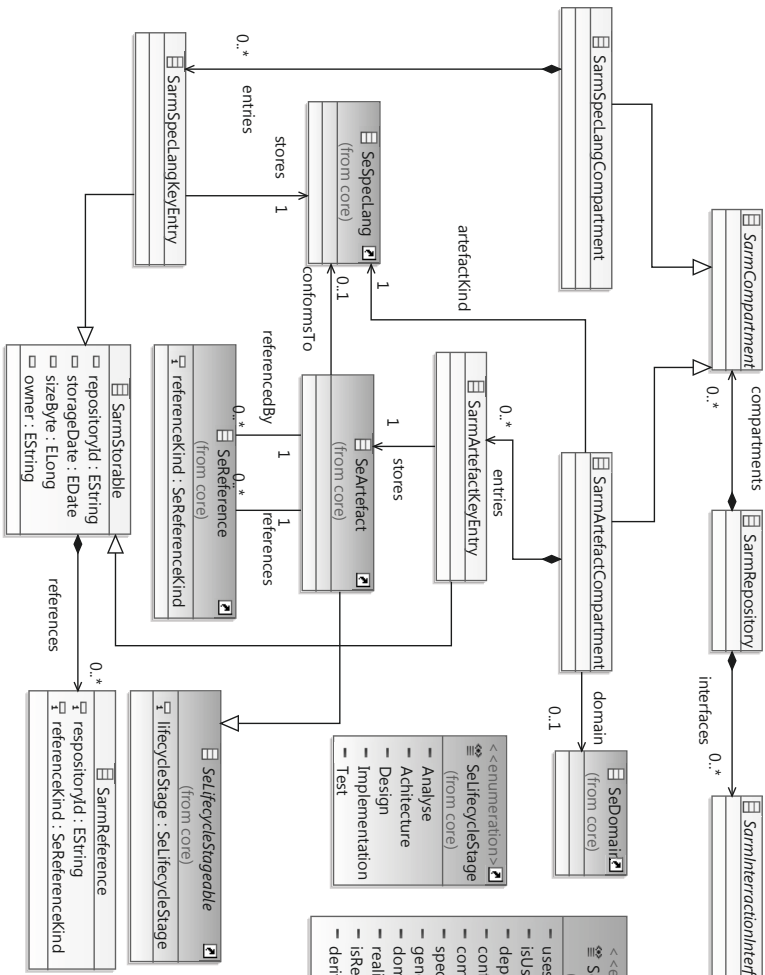

Fig. 4. Repository Specification Language -Overview

An SeLifecycleStage defines an enumeration to the development life-cycle stage in which the artifact will be used. In our study, we focus on S&D pattern models. In this context, we use the pattern classification of Riehle and Buschmann [4,19].

- SarmCompartment. Is used for the categorization of the stored artifacts. We have identified two main kinds of compartments: (1) SarmSpecLangCompartment to store the specification languages (SeSpecLang) of the modeling artefacts (SEPM), and (2) SarmArtefactCompartment to store the modeling artefacts (S&D pattern models).
- SeReference. This link will be used to specify the relation between patterns with regard to domain and software life-cycle stage in the form of a pattern language. For instance, a pattern at a certain software life-cycle stage *uses* another pattern at the same/or at different software life-cycle stage. The enumeration SeReferenceKind contains examples of these links.
- SarmStorable. Is used to define a set of characteristics of the modeling artifacts, mainly those related to its storage. We can define: *RepositoryID*, *StorageDate*, *SizeByte*, etc.... In order to keep the structure of pattern language as the set of patterns and their links for a certain domain, the concept SarmStorable includes a list of references (SarmReference).

4.2 Pattern Specification Language (SEPM)

The System and software Pattern Metamodel (SEPM), as depicted in Fig. 5, is a metamodel defining a new formalism for describing patterns. Note, however, that our proposition is inspired from GoF [20] specification, which we deeply refined in order to fit with the non-functional needs. The principal classes of the metamodel are described with Ecore notations in Fig. 5. In the following, we detail the meaning of principal concepts used to edit a pattern.

- SepmPattern. This block represents a modular part of a system representing a solution of a recurrent problem. It specializes the conceptual SeArtifact. An SepmPattern is defined by its behavior and by its provided and required interfaces. An SepmPattern may be manifested by one or more artifacts, and in turn, that artifact may be deployed to its execution environment. The SepmPattern has attributes [20] to describe the related recurring design problem that arises in specific design contexts.
- SepmInternalStructure. Constitutes the implementation of the solution proposed by the pattern. Thus the *InternalStructure* can be considered as a white box which exposes the details of the pattern.
- SepmInterface. A pattern interacts with its environment with *Interfaces* which are composed of *Operations.* We consider two kinds of interface:
  (1) SepmExternalInterface for specifying interactions with regard to the integration of a pattern into an application model or to compose patterns, and
  (2) SepmTechnicalInterface for specifying interactions with the platform.

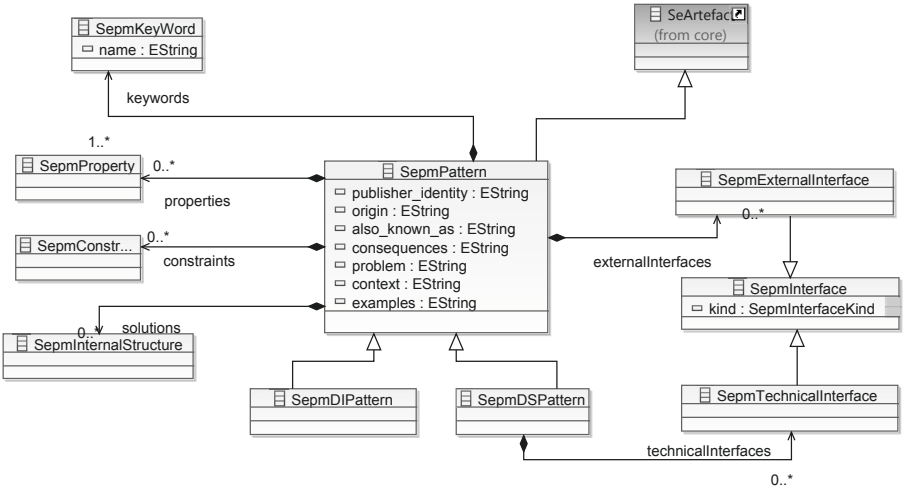

Fig. 5. Pattern Specification Language -Overview

– SepmProperty. is a particular characteristic of a pattern related to the concern dealing with and dedicated to capture its intent in a certain way. Each property of a pattern will be validated at the time of the pattern validation process and the assumptions used will be compiled as a set of constraints which will have to be satisfied by the domain application. Security attributes [21] such as *Confidentiality* and *Availability* are categories of S&D properties.

5 MDE Tool-Chain

Once these specification languages have been defined, it is possible to develop a repository in which modeling artifacts specifications and instances are stored. There are several Domain Specific Modeling Languages (DSML) [7] environments available. In our context, we use the Eclipse Modeling Framework (EMF) [22] open-source platform. Note, however, that our vision is not limited to the EMF platform. Using the proposed metamodels, ongoing experimental work is done under the hood of *semcomdt*[3] (IRIT's editors and platform as Eclipse plug-ins), testing the features of:

(1) *Gaya G* for the repository structure and API conforming to *SARM*,
(2) *Arabion(A)* for specifying patterns and documentation generation conforming to *SEPM*, and
(3) *Deposit* and *Retrieval* for repository access.

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

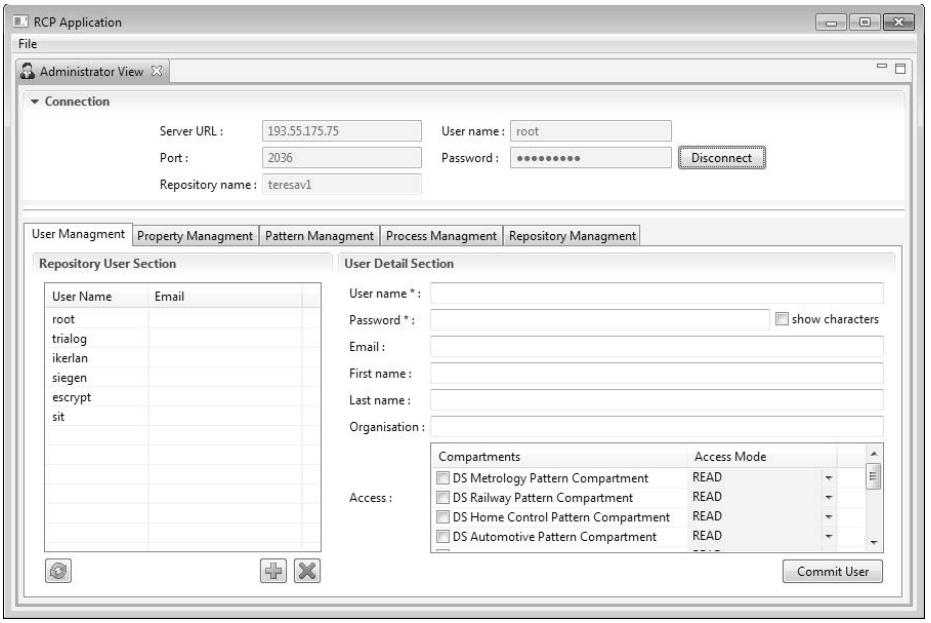

Fig. 6. Repository Implementation

5.1 Repository - Gaya

The implementation of Gaya is based on the SARM metamodel and the repository of S&D pattern structure presented in Sect. 4 on one hand and on Eclipse CDO[4] on the other hand. Repository management is provided via the Gaya tool. Gaya offers repository management with facilities such as pattern lookup, removal, sorting, exporting and categorization. We offered these facilities through a set of dialogs. The main dialog is shown in Fig. 6.

5.2 Pattern Designer - Arabion

The pattern designer called *Arabion* is an EMF tree-based editor for specifying S&D patterns. The design environment is presented in Fig. 7. There is a design palette on the right, a tree view of the project on the left and the main design view in the middle. The design palette is updated regarding the development stage to display suitable design entities for building patterns. These entities are *technical interfaces* and *resource properties*.

In our example, the *SecurityCommunicationLayer@DetailedDesign* uses the *HMAC* mechanism. The call of the method *send()* of the Sender calls internally *generateAH()* to prepare an appropriate authentication header for the data.

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

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

Moreover, Arabion includes conformance validation tool used to guarantee design validity conforming to the pattern metamodel. In our example, the Secure Communication pattern model can be validated, where a violation of a metamodel construct will yield an error message.

5.3 Pattern Deposit

Pattern publication is triggered by running the *Publication* tool. When executed, the pattern will be stored in the repository following the pattern designer's profile (compartment). The tool uses the *Gaya4Pattern API*, for publishing to the repository. Note, however that the deposit tool requires the execution of the validation tool to guarantee design validity.

5.4 Pattern Retrieval

The tool allows the search/selection of patterns which are used during a system development process. For instance, as shown in the right part of Fig. 8, the tool provides a set of facilities to help the selection of appropriate patterns. The results are displayed in search results tree as System, Architecture, Design and Implementation patterns. For example, the right part of Fig. 8 shows a pattern at design level targeting the *Confidentiality* S&D property[5], named communication and has a keyword *secure*. The tool includes features for exportation and instantiation. In our case, we select the *Secure Communication* pattern for instantiation (see the left part of Fig. 8).

6 Application of the PBSE Methodology to a Case Study

To illustrate our approach we have an industry control application from the railway domain called *Safe4Rail* acting as a TERESA case study. Our goal is also to assess whether the PBSE addresses the practical needs when modeling the trusted embedded system application of a realistic system and whether it can provide significant benefits in terms of reducing modeling effort and error-proneness.

The application is in charge of the emergency brake of a railway system. Its mission is to check whether the brake needs to be activated. Their implementation mainly depends on the safety level to meet, but also on the type and the

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

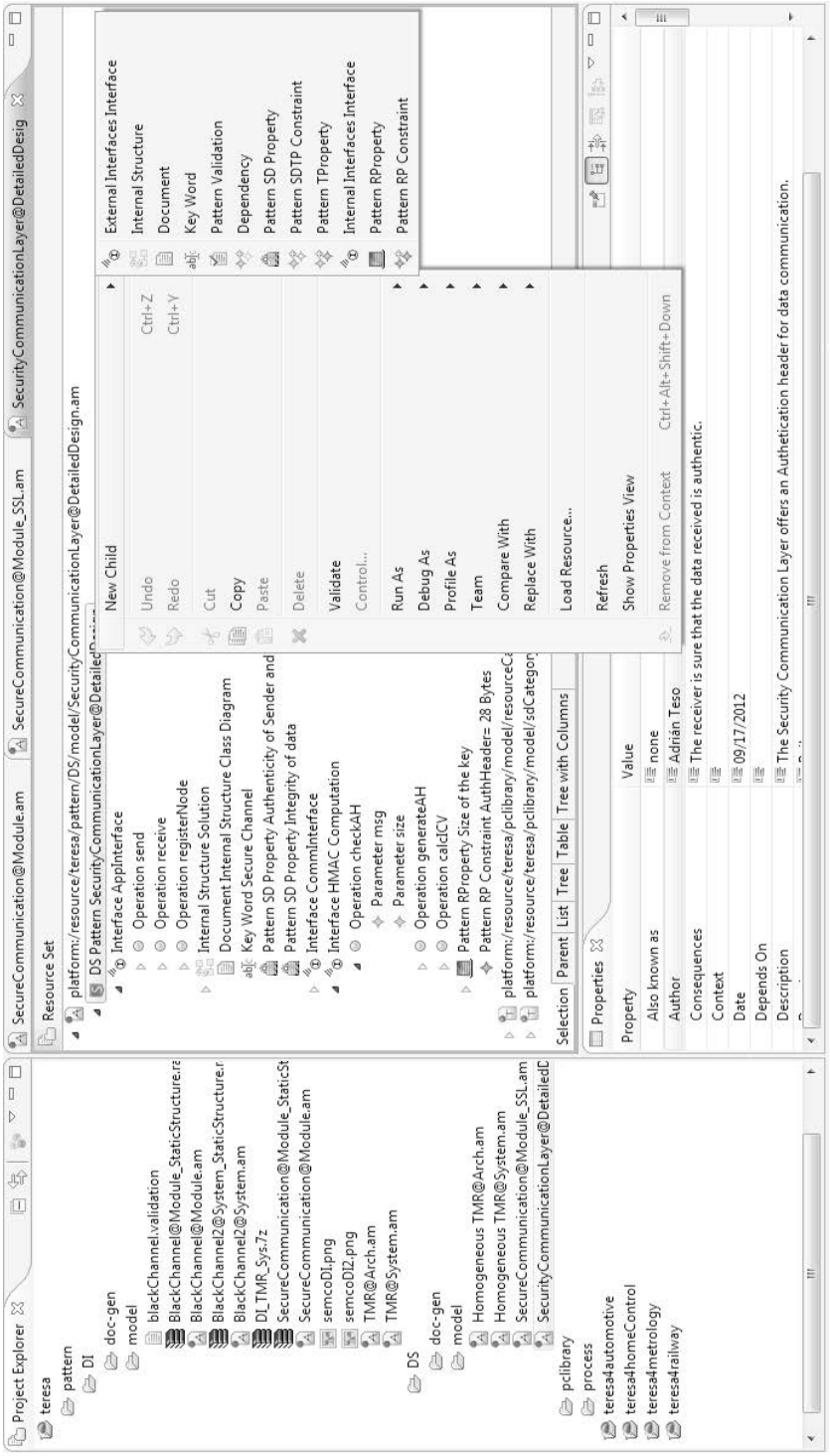

Fig. 7. Secure Communication Pattern at Design level

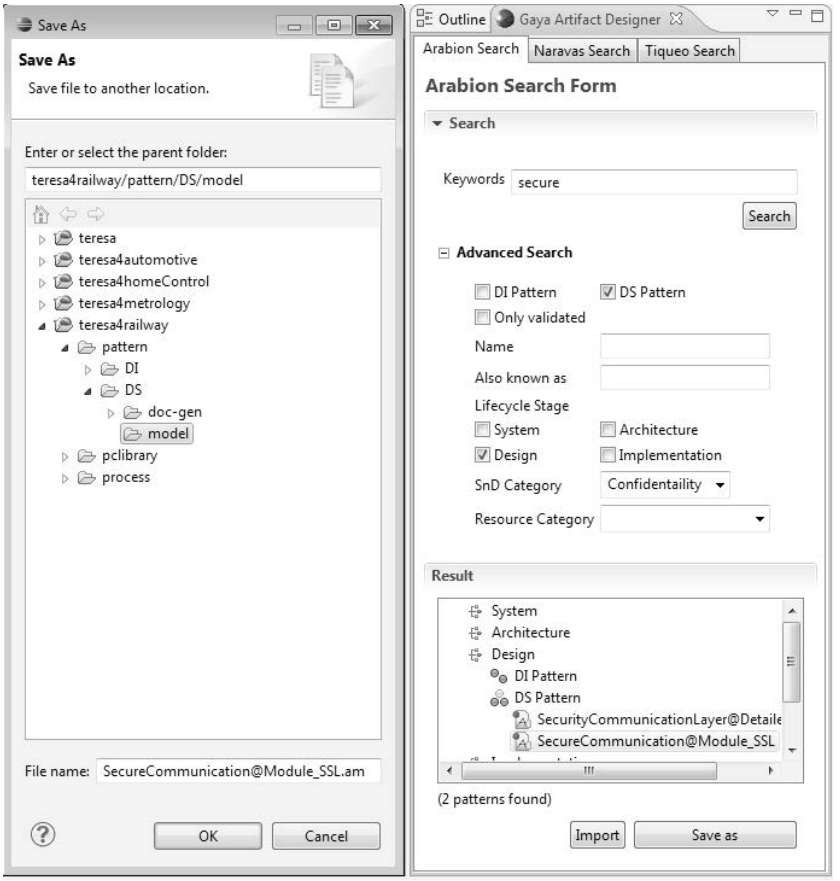

Fig. 8. Pattern Instantiation

number of sensors and actuators involved. These considerations greatly influence how each product is to be implemented (e.g, the number of channel redundancy, the diversity of the channels,. . .). In this case, SIL4 level is targeted. A number of design techniques from S&D are used, namely redundancies, voting, diagnostics, secure and safe communications. A very strict engineering process was followed, where specific activities were performed in order to achieve certification using the presented approach.

6.1 A Model-Based Repository of S&D Patterns Structure Model

The railway domain analysis led to identification of a set of patterns for the Safe4Rail application. We used Arabion to design these patterns and then the Deposit tool to store them in the repository. Figure 9 depicts an overview of the railway S&D pattern language.

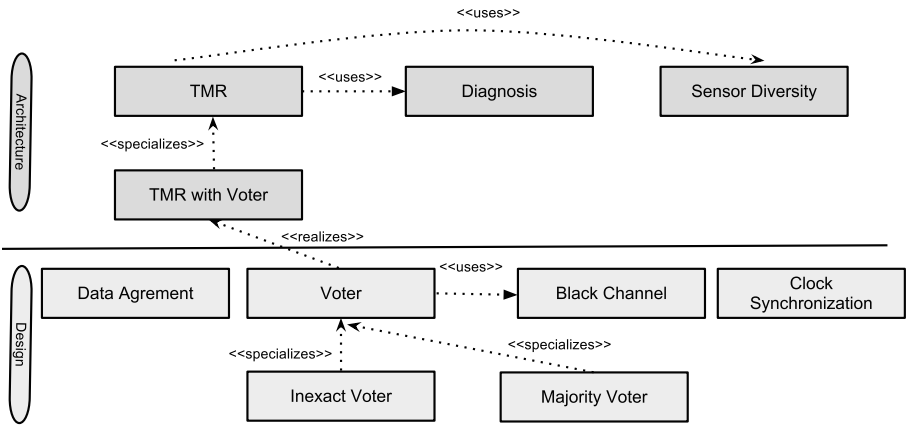

Fig. 9. Railway Pattern Language - Overview

6.2 System Developer View Point: Reuse of Existing Artifacts

Here, we examine the process flow for the example following the design process of Sect. 3.4. Once the requirements are properly captured and imported into the development environment [6], the process can be summarized with the following steps:

Activity A2: Develop architecture model of a system. The analysis of the requirements (A3) results in the needs of an architectural pattern for redundancy. Thus, activity A4 is the instantiation of S&D patterns from the repository using the repository access tools. The running of the Retrieval tool using keywords *Redundancy* and *SIL4*, suggests to use a *TMR* pattern at architecture level. In addition, some diagnosis techniques imposed by the railway standard are suggested, thanks to the repository organization for the railway application domain (see Fig. 9). Finally, at architecture level, we will integrate (A5) the following patterns:

(1) TMR (searched by the System Architect),
(2) Diagnosis techniques (suggested by the tool) and
(3) Sensor Diversity (searched by the System Architect).

Activity A7: Develop design model of a system. This activity involves the development of the design model of the system. The analysis of the requirements (A8), the architecture model and the identified architectural patterns will help during the instantiation activity (A9) of the design phase. Based on the selected patterns, the repository may suggest related or complementary patterns.

[6] Rhapsody is used by Ikerlan Center engineers.

For instance, if the TMR has been integrated, the following patterns may be proposed for the design model iteration: (1) Data Agreement, (2) Voter, (3) Black Channel and (4) Clock Synchronization.

7 Assessment

This section provides a preliminary evaluation of the approach along TAM (Technology Acceptance Model) and concerns the methodology as well as the tools (Arabion and Gaya). We have identified a set of measures to evaluate the usage of the models and the user-friendliness of the tools. Eleven TERESA members participated. The study was divided into three tasks. Before they started, a general description of the aim of the study was given (30'). Some running examples were introduced to them. After these two tasks, achieved during the TERESA MDE workshop in Toulouse (April 2012), a 6-months evaluation was conducted. All the subjects were already familiarized with MDE, S&D patterns and Eclipse. The procedure includes four tasks: SEMCO plug-in installation, pattern development, patterns instantiation and patterns integration.

We asked participants to give scores from 1 to 5 (5 is the best). We first evaluated the perceived usefulness of the solution itself (items 1-4). Next, we focus on the ease of the solution (items 5-6). We want also to measure the compatibility of the solution with existing environments. (items 7-9). Finally, we wanted to measure the willingness to use the approach in the future in the related activities (items 10-12). These scores indicate the degree of satisfaction of the users and provides a feedback to us in order to enhance our specification languages and the tool suite. The following table depicts an overview of the results of our experiment.

Item	**Mean**	**St. Deviation**
1. Design quality	3.5	0.4
2. Model completeness	4	0.3
3. Documentation and artifact generation readability	4	0.89
4. Effort spent on development	4.5	0.16
5. Model understandability	3.5	0.475
6. Effectiveness	3.5	0.6
7. Integration with other solutions	4	0.86
8. Standards compliance	4	0.2
9. Cost of adoption	3.5	0.48
10. Use the approach in the future	4.10	0.68
11. Exchange the approach in the future	3.60	0.56
12. Customize some of the proposed plug-ins in the future	3.60	0.76

Fig. 10. Satisfaction Results

8 Conclusion

We proposed a methodology and an MDE tool-chain to support the specifications and the packaging of a set of S&D patterns, in order to assist the developers of trusted applications for resource constrained embedded systems.

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

References

1. Zurawski, R.: Embedded Systems. CRC Press Inc. (2005)
2. Ravi, S., et al.: Security in embedded systems: Design challenges. Transactions on Embedded Computing Systems (TECS) 3(3), 461–491 (2004)
3. 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)
4. Riehle, D., Züllighoven, H.: Understanding and Using Patterns in Software Development. Theory and Practice of Object Systems 2(1), 3–13 (1996)
5. Serrano, D., Mana, A., Sotirious, A.-D.: Towards Precise Security Patterns. In: 19th International Conference on Database and Expert Systems Application, DEXA 2008, pp. 287–291. IEEE Computer Society (2008), http://doi.ieeecomputersociety.org/10.1109/DEXA.2008.36
6. Crnkovic, I., et al.: Component-Based Development Process and Component Lifecycle. In: Proceedings of the International Conference on Software Engineering Advances, ICSEA 2006, p. 44. IEEE Computer Society (2006)
7. Gray, J., et al.: Domain-Specific Modeling. Chapman & Hall/CRC (2007)
8. Daniels, F., Kim, K., Vouk, M.A.: The reliable hybrid pattern – a generalized software fault tolerant design pattern. In: Pattern Language of Programs, PLoP 1997 (1997), http://hillside.net/plop/plop97/Proceedings/daniels.pdf
9. Tichy, M., Schilling, D., Giese, H.: Design of self-managing dependable systems with UML and fault tolerance patterns. In: Proceedings of the 1st ACM SIGSOFT Workshop on Self-Managed Systems, WOSS 2004, pp. 105–109. ACM (2004)
10. Yoder, J., Barcalow, J.: Architectural patterns for enabling application security. In: Pattern Languages of Programs, PLoP 1998 (1998), http://hillside.net/plop/plop97/Proceedings/yoder.pdf
11. Schumacher, M.: Security Engineering with Patterns. LNCS, vol. 2754. Springer, Heidelberg (2003)
12. Jürjens, J.: UMLsec: Extending UML for secure systems development. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 412–425. Springer, Heidelberg (2002)

13. Hatebur, D., Heisel, M., Schmidt, H.: A security engineering process based on patterns. In: Proceedings of the 18th International Workshop on Database and Expert Systems Applications, DEXA 2007, pp. 734–738. IEEE Computer Society (2007)
14. Halkidis, S.T., Chatzigeorgiou, A., Stephanides, G.: A qualitative analysis of software security patterns. Computers & Security 25(5), 379–392 (2006)
15. Konrad, S., et al.: Using security patterns to model and analyze security requirements. In: Requirements Engineering for High Assurance Systems, RHAS 2003, pp. 13–22. Software Engineering Institute (2003)
16. Di Giacomo, V., et al.: Using security and dependability patterns for reaction processes. In: 19th International Workshop on Database and Expert Systems Application, DEXA 2008, pp. 315–319. IEEE Computer Society (2008)
17. Fernandez, E.B., et al.: Using security patterns to develop secure systems. In: Software Engineering for Secure Systems, pp. 16–31. Information Science Reference (2011)
18. Alexander, C., Ishikawa, S., Silverstein, M.: A pattern language – towns, buildings, construction, vol. 2. Oxford University Press (1977)
19. Buschmann, F., Henney, K., Schmidt, D.C.: Pattern-oriented Software Architecture, vol. 4. John Wiley & Sons (2007)
20. Gamma, E., et al.: Design patterns – Elements of reusable object-oriented software. Addison-Wesley (1995)
21. Avizienis, A., et al.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1(1), 11–33 (2004)
22. Steinberg, D., et al.: EMF: Eclipse Modeling Framework 2.0. Addison-Wesley (2008)