Towards Tool Support for Pattern-Based Secure and Dependable Systems Development

Brahim Hamid, Adel Ziani, Jacob Geisel

Towards Tool Support for Pattern-Based Secure and Dependable Systems Development

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

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

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

ABSTRACT

In our work, we promote a new discipline for secure and dependable system engineering using a pattern as its first class citizen: Pattern-Based System Engineering (PBSE). Therefore, PBSE addresses two kind of processes: the one of pattern development and the one of system development with patterns. To interconnect these two processes we promote a structured model-based repository of patterns and their related property models.

This paper presents the SEMCO MDE Tool Suite development status conducted in the context of the FP7 TERESA project aiming to support the automation of building, storing and processing reusable artifacts (S&D patterns and property models). This tool promotes the PBSE methodology in the domain of assistance to the trusted embedded system engineering.

A video tutorial presenting the SEMCO MDE Tool Suite is provided under: `http://www.semcomdt.org/semco/demo/video_semco/toolsuite/ToolSuiteIRIT.mp4`

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Design Tools and Techniques—*Design,Software Architectures,Reusable Software*

General Terms

design

Keywords

Embedded Systems, Security, Dependability, Repository, Pattern, Metamodel, Model-Driven Engineering.

1. INTRODUCTION

The software of embedded systems 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) 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 our previous work [1], we studied pattern modeling frameworks [2, 3] and we proposed methods to model security and dependability aspects in patterns 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 S&D patterns are involved. We promote a new discipline for system engineering using a pattern as its first class citizen, towards meeting our wider objective: Pattern-based System Engineering (PBSE). PBSE addresses challenges similar to those studied in software engineering. Closely related to our vision is the Component Based Software Engineering (CBSE) [4]. 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 an Model-Driven Engineering Tool-chain supporting the PBSE methodology, and hence to assist the developers of secure and dependable systems. The framework is centered around a model-based repository of S&D patterns and models providing operational repository, tools for managing, tools for populating and accessing the repository. At the core of the framework is a set of Domain Specific Modeling Languages (DSML) [5, 6] that allow modeling S&D patterns, property models and repository structure.

The work is conducted in the context of a framework called *SEMCO*[1] for System and software Engineering for embedded systems applications with Multi-COncerns support. SEMCO is a federated modeling framework built on an integrated repository of metamodels to deal with system engineering. The end-user part of such a framework is an integrated repository of modeling artifacts to be used in order (1) to promote engineering separation of concerns, (2) to support multi-concerns, (3) to use *patterns* to embed solutions of engineering concerns and (4) to support

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

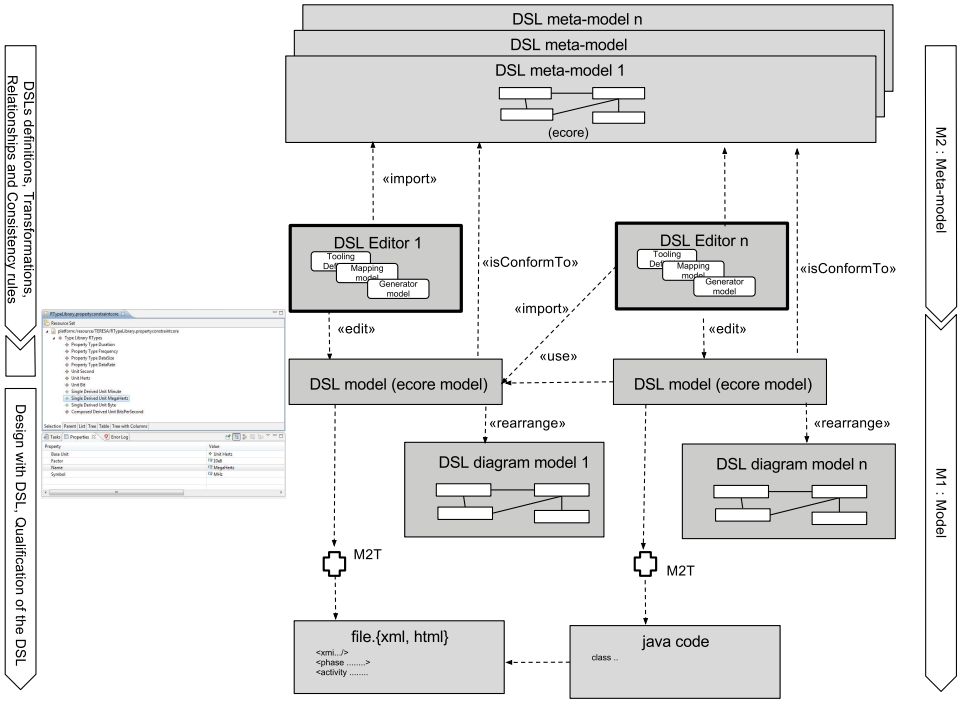

Figure 1: SEMCO DSL building process and artifacts

multi-domain specific processes. This project is three-folded: providing repository of modeling artifacts, tools to manage these artifacts, and guidelines to build complete engineering systems.

Domain Specific Modeling Languages (DSML) [6] has recently increased in popularity to cover a wider spectrum of concerns. As we shall see, such a process reuses many practices from Model-Driven Engineering. For instance, metamodeling and transformation techniques. SEMCO foundation is a federated DSLs processes working as a group on how relevant each one is to the key concern. As shown in Figure 1, a DSL process [2] is divided into several kinds of activities: DSL definition, transformation, consistency and relationships rules, design with DSL and Qualification. The three first activities are achieved by the DSL designer and the two last activities are used by the final DSL user.

There are several DSM environments available. In our context, we use the Eclipse Modeling Framework (EMF) [7] open-source platform to support such a process and to create our tool suite. Note, however, that our vision is not limited to the EMF platform.

The rest of this paper is organized as follows. An overview of the modeling approach we proposed including a set of DSLs is presented in Section 2. Then, Section 3 presents the architecture and the functionality of the tool-chain. Section 4 presents in detail the implementation of the tools composing the chain. In Section 5, we provide a first feedback on the tool-suite we propose. Finally Finally, Section 6 concludes this paper with a short discussion about future works.

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

2. OVERVIEW OF THE DSMLS

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

2.1 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 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. 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.
- 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 artifacts (SEPM and GPRM), and (2) SarmArtefactCompartment to store the modeling artifacts (S&D pattern and property 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).

2.2 Generic Property Modeling Language

The metamodel of property [8] (GPRM) captures the common concepts of the two main concerns of trusted RCES applications: *Security*, *Dependability* and *Resource* on the one hand and *Constraints* on these properties on the other hand. The libraries of properties and constraints includes units, types, categories and operators. For instance, security and dependability attributes [9] 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 design of the pattern (see next sections) we define the properties and the constraints using these libraries.

2.3 Pattern Specification Language

The System and software Pattern Metamodel (SEPM) [1] is a metamodel defining a new formalism for describing patterns. Note, however, that our proposition is inspired from GoF [10] specification, which we deeply refined in order to fit with the non-functional needs. 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 [10] (name, problem, context, ...) 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.

- 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 [9] such as *Confidentiality* and *Availability* are categories of S&D properties.

3. IMPLEMENTATION ARCHITECTURE

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

3.1 Tool-suite Architecture

TERESA provides three integrated sets of software tools: (i) *Tool set A* for populating the repository, (ii) *Tool set B* for retrieval from the repository and (iii) *Tool set C* for managing the repository. As shown in Figure 2, thanks to UML component diagram the tool-suite is composed of:

- *Gaya (G).* for the repository structure and interfaces conforming to *SARM*,

- *Tiqueo (T).* for specifying models of S&D properties conforming to *GPRM*,

- *Arabion (A).* for specifying patterns conforming to *SEPM*,

- *Admin.* for the repository management,

- *Retrieval.* for the repository access.

3.2 Tool-suite Functionality

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

3.2.1 Repository Set-up

GAYA is a repository platform to store the modeling artifact specifications and instances through the APIs. The server part is responsible for managing and storing the data, and provides a set of features to interact with the repository content. As shown in Figure 2, the server part is composed of two components: (1) *GayaServer* providing the implementation of the common API and (2) *GayaMARS*

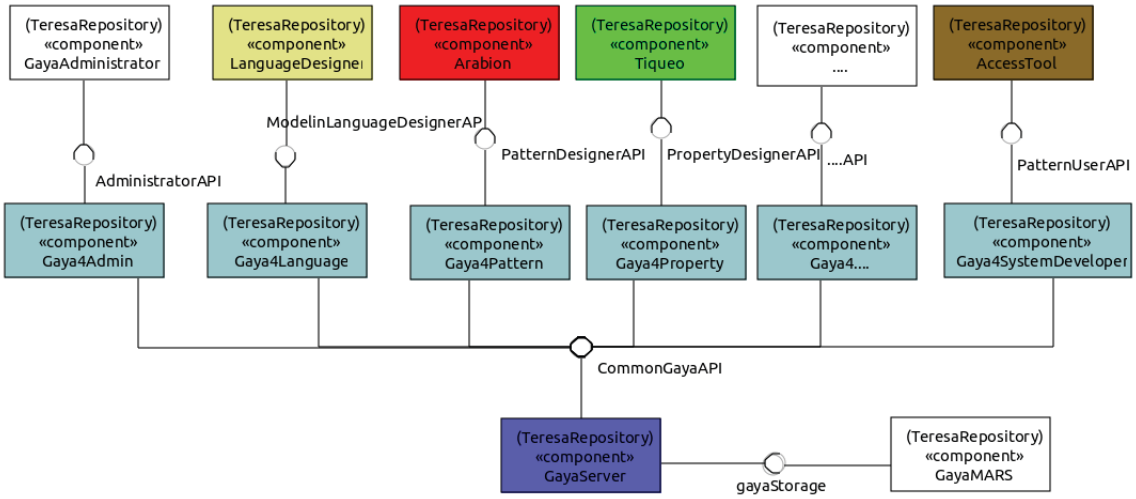

Figure 2: The tool suite architecture

providing the storage mechanisms. The client part is responsible for populating the repository and for using its content providing APIs interfaces for applications, such as depicted in Figure 2, in order to populate, access and to manage the repository. For instance, *Gaya4Pattern* (implements the API4PatternDesigner), *Gaya4Property* (implements the API4PropDesigner), *Gaya4Admin* (implements API4Admin) and *Gaya4SystemDeveloper* (implements the API4PatternUser).

3.2.2 Repository Populating - Design Tools

The property designer (Tiqueo), to be used by a *property designer*, provides features for specifying models of properties and constraints. In addition, Tiqueo provides some features to create a library for reusable objects, like the types and units which allows us to use the libraries in a domain independent manner. Furthermore, Tiqueo includes mechanisms to validate the conformity of the property and constraint library to the GPRM metamodels and to publish the results into the Gaya repository using the *Gaya4Property* API.

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

3.2.3 Repository Managing

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

3.2.4 Repository Accessing (Retrieval)

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, domain independent vs. domain specific and property categories. The results are displayed in search result tree as System, Architecture, Design and Implementation patterns. The Tool includes features for exportation and instantiation as dialogues targeting domain specific development environment. Moreover, the tool includes dependency checking mechanisms. For example, a pattern can't be instantiated, when a property library is missing, an error message will be thrown.

4. IMPLEMENTATION DETAILS

Using the proposed metamodels and the Eclipse Modeling Framework (EMF) [7], ongoing experimental work is done with *semcomdt* [3] (SEMCO Model Development Tools, IRIT's editor and platform plugins).

4.1 CDO Repository Implementation

Our approach, relying on an MDE based techniques to build a set of DSLs and thus in our context supporting automated model-based repository building, such as visualized in Figure 3. Then, we provide the environment for the use of the resulted repository through APIs.

The models specifying the structure of the repository and the APIs are built through an EMF tree-based editor implementing the SARM metamodel, as shown in the top part of Figure 3. The resulting Ecore model is then used as input for the model to text transformations in order to generate the repository and APIs software implementation artifacts targeting the CDO platform [11] (as seen in the mid part of Figure 3).

The structure of the repository is derived from the repository structure model and implemented using Java and the Eclipse CDO Server technology. The server part of the

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

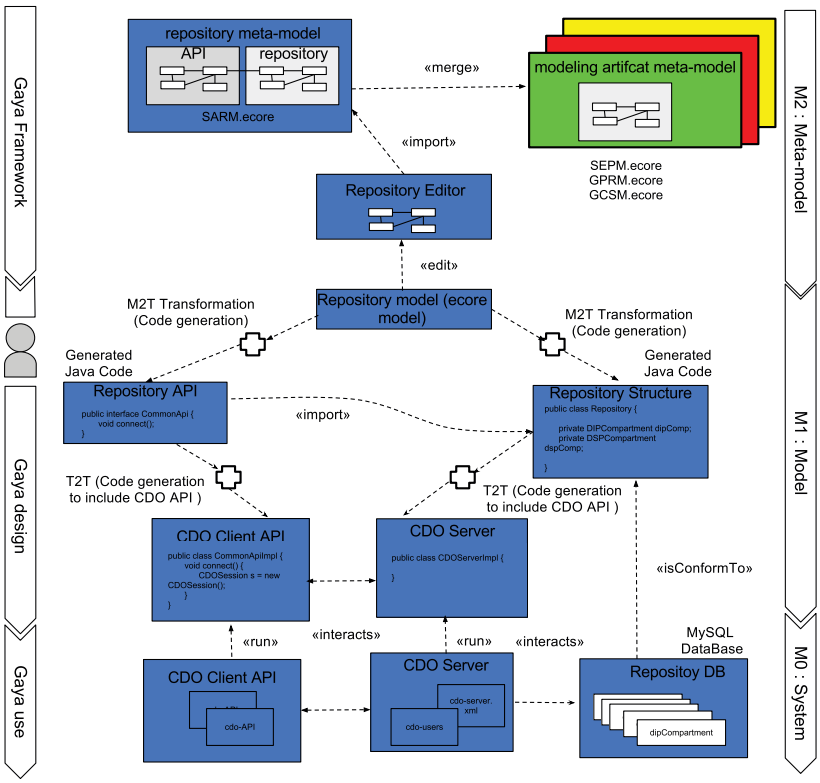

Figure 3: The Model-based repository building process

repository is provided as an Eclipse plugin that will handle the launch of a CDO server defined by a configuration file. The repository client APIs are derived from the repository APIs model and then implemented as CDO clients. In our example, the repository interfaces model are visualized in Figure 4. We specified a set of functions and the data structure of their parameters in the form of UML class diagram. The implementation is based firstly on the automatic code generation from the APIs model. In our development environment, the generated Java code defines the different interfaces and functions provided by the repository APIs. The skeleton of the APIs implementations are then completed manually based on CDO technology. As the CDO server, the CDO clients are provided as Eclipse plugins (as shown in the down part of Figure 3).

4.2 Design Tools

As shown in Figure 5, we used the Eclipse Modeling Framework (EMF) to support such a process and to create our tool suite design editors: (1) we create the model's Ecore file (MM) and then (2) we use the code generation techniques of EMF as a transformation engine to build DSLs editors. That is, we derived the EMF Generator Model (genmodel) file based on the model's Ecore file to make use of the code generation techniques of EMF. Then we generated the model, the editor and the test code, as well as the different Eclipse plugins, using these two files (Ecore and genmodel). The generated editor code was modified to limit the user actions on the ones needed and to enhance user experience (e.g. modifying the name of some concepts). The tool provides facilities for editing modeling artifacts instances in a domain independent manner (DIM). Then the user can refine the guidelines for domain-specific application (DSM). Further, using EMF features, we added the metamodel conformance function to the editor (Model Checker) and code/documentation generation (Generator).

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

We applied the DSL process to build the property designer and the pattern designer: TIQUEO is an EMF generated tree-based editor for specifying models of properties and constraints, and ARABION is an EMF generated tree-based editor for specifying domain independent and domain specific patterns. Then, we applied the code generation strategy for HTML generation for both pattern and property models.

For a pattern, the design environment is presented in Figure 6. 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. In our example, the *SecurityCommunicationLayer@DetailedDesign* pattern uses the *HMAC* mechanism. The call of the method *send()* of the Sender calls internally

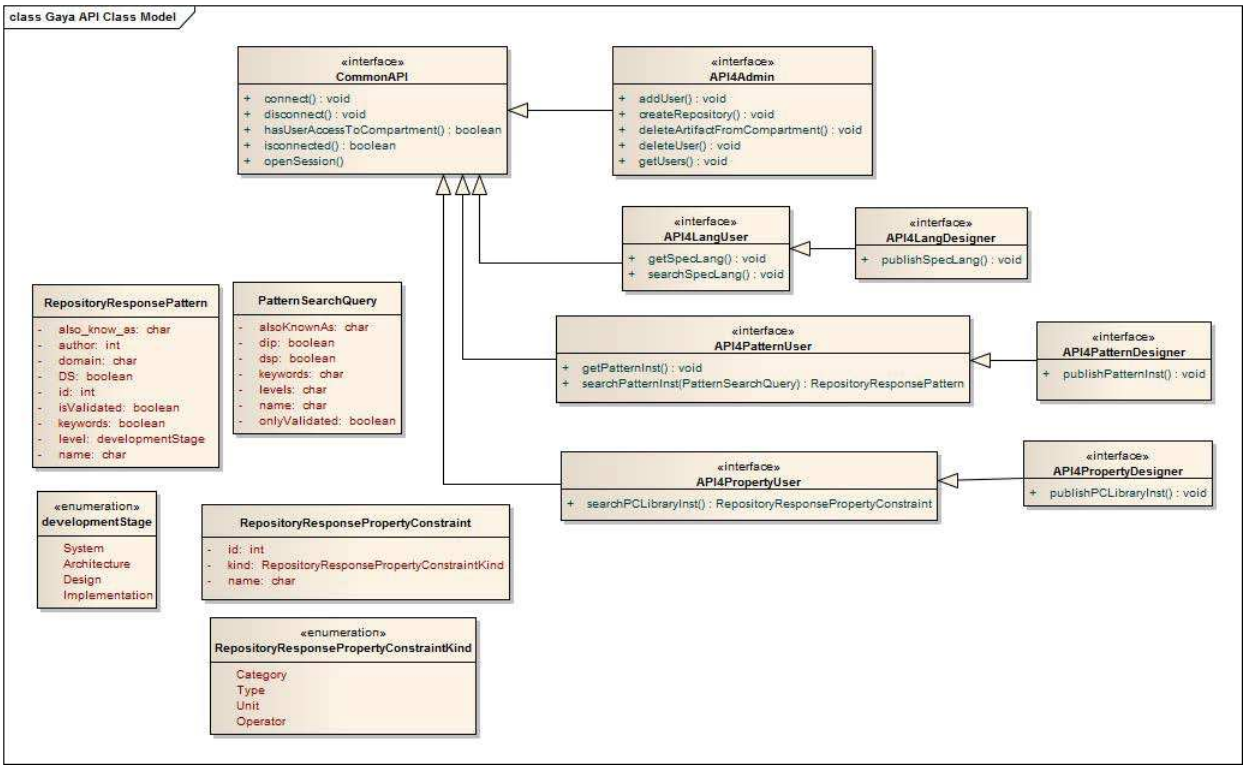

Figure 4: The Repository Interfaces and Classes

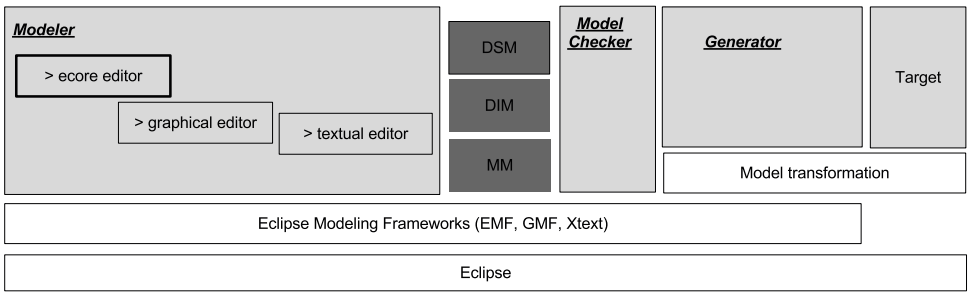

Figure 5: Overview of the Semco tool suite architecture

to *generateAH()* to prepare an appropriate authentication header for the data. Once this header is appended to the message it is sent by the communication channel. On the 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 is discarded. The operations *generateAH()* and *checkAH()* are provided through an internal interface called *HMAC Computation*.

To type the category of an S&D property, the user has to create a reference to the library as a resource. As a prerequisite, the designer uses the *Retrieval* tool (see subsection 4.2.2) to search and then to upload the appropriate library in its environment.

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

4.2.1 Repository Managing (Admin)

We provide software, as a Java based GUI application, called *GayaAdmin* to manage relationships among S&D patterns specifications, and between S&D patterns and their related property models. For instance, as visualized in Figure 7, a pattern may be linked with other patterns and associated with property models using a predefined set of reference kinds such as those proposed in the SARM metamodel. Moreover, we support basic features such as artifact management and user management. *GayaAdmin* uses the *Gaya4Admin* API.

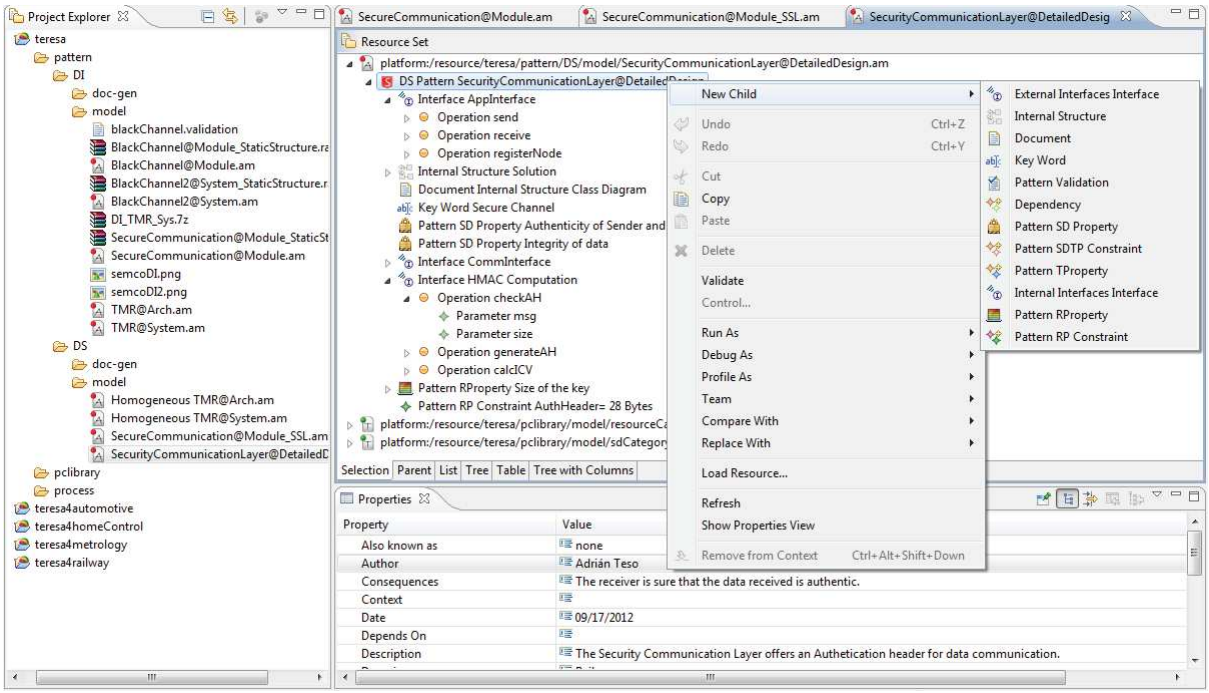

Figure 6: Pattern designer -Arabion

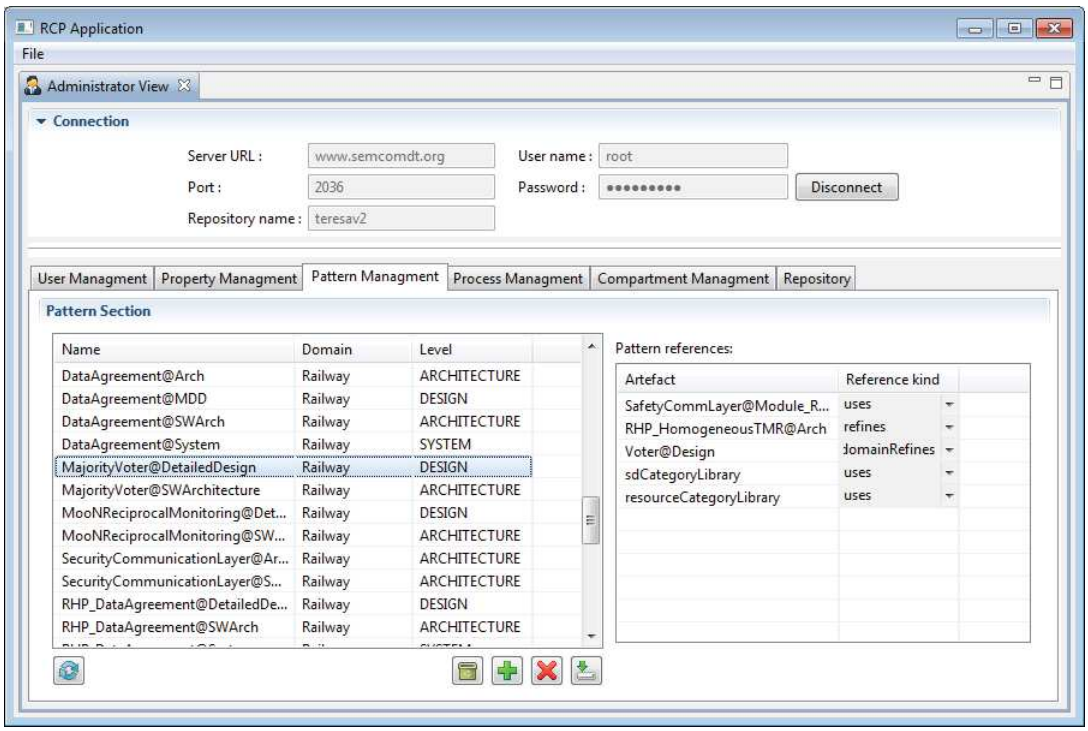

Figure 7: Repository organization

4.2.2 Repository Accessing (Retrieval)

To access the repository, we are building an Eclipse plugin application, which uses the Gaya4SystemDeveloper API for the search/selection/sorting of the patterns. The Tool includes features for exportation and instantiation as dialogues, mainly those based on model transformation techniques to adapt the model of the pattern to the target development environment (for example Rhapsody UML).

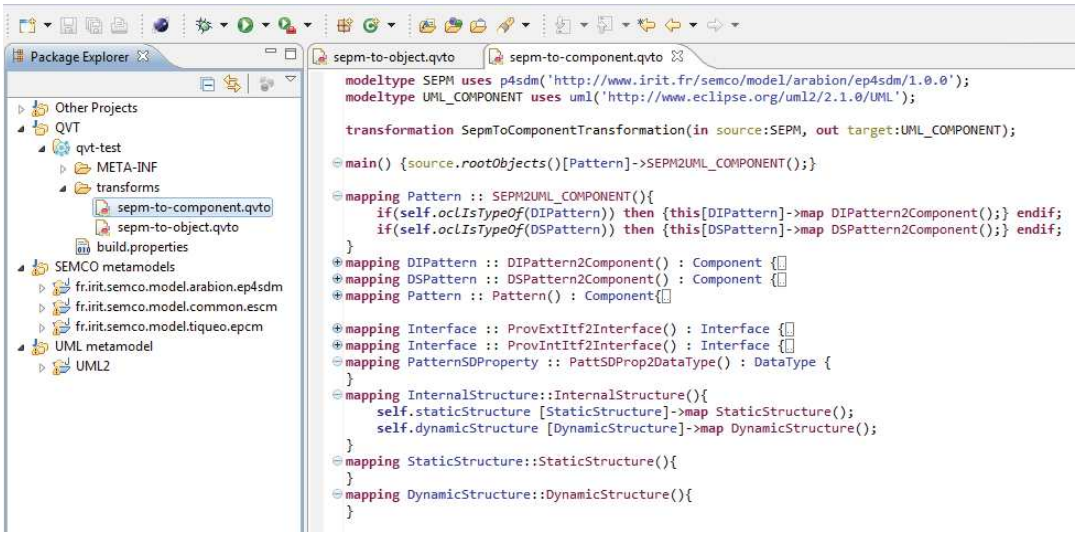

Figure 8: Mapping rules from SEPM to UML Component using QVT

A transformation (in our case implemented with the OMG standard QVT – Query/View/Transformation) synchronizes the pattern model between the SEPM pattern model and UML model. For instance, we propose a process within a set of model transformation rules to target UML object and component architectures. In Figure 8, we show an overview of a set of transformation rules using QVT [12] under EMF. SEPM and UML are specified using Ecore metamodel to be used as model types inputs.

5. ASSESSMENT

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

In the context of the TERESA project[4], we evaluated the tool-chain to build two demonstrators combining MDE and a model-based repository of S&D patterns and their related property models: (1) Railway *Safe4Rail* application in charge of the emergency brake of a railway system and (2) Metrology *SmartMeterGateway* application in charge of connecting Smart Metering devices. Figure 9 shows a set of roles and the environment of the integrated modeling, storage and system development process based on our vision and approach.

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

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, though some did not know some of the companion plugins (e.g. Acceleo). Hence, the generation of documentation was not part of the evaluation. The procedure includes four tasks: SEMCO plug-in installation, property models development, pattern development and patterns instantiation.

- *Effectiveness.* Figure 10 shows a table providing the fulfillment for the five tasks. One subject had problems in using UML editors (Rhapsody or Papyrus) for pattern integration, and hence, he was excluded from the rest of the experiment.

Item	Frequency	%
Task 1. Plugin installation	3.5	100
Task 2. Property Model development	5.5	100
Task 3. Pattern development	10	100
Task 4. Pattern instantiation	11.2	100

Figure 10: Effectiveness Results

- *Productivity.* Productivity is measured as the number of minutes required for each task (only for those that successfully completed the first task). As shown in Figure 11, SEMCO plugins installation took between 10 and 15 minutes, with a mean of 12; property model development took between 20 and 60 minutes, with a mean of 42.5 minutes; pattern development took between 40 and 60 minutes, with a mean of 53 minutes; pattern instantiation took between 10 and 30 minutes, with a mean of 19 minutes and finally.

Item	Mean (minutes)	St. Dev.
Task 1. Plugin installation	12	2.8
Task 2. Property Model development	42.5	11.5
Task 3. Pattern development	53	8.4

Figure 11: Productivity Results

- *Satisfaction.* Satisfaction is the capability of the software product to satisfy its users. In this case, the

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

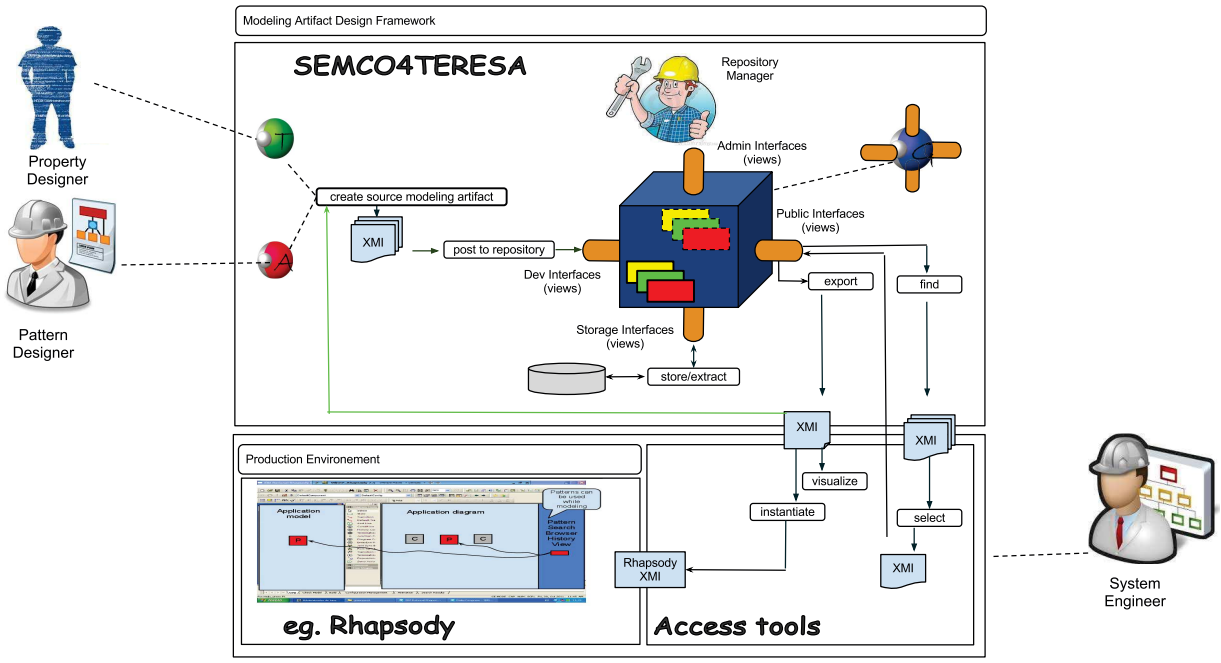

Figure 9: Integrated Modeling/Configuration/Assembly Process

product is the repository of S&D patterns engine, and its ability to develop a trusted RCES application. We asked participants to give scores from 1 to 5 (5 is the best) and comments. We focus on the tool-suite as a mean to build the modeling artifacts. We separately collected the satisfaction along the four tasks (items 1-7). Finally, we want also to measure the willingness to use repository of modeling of S&D patterns in the future in the related activities (items 8-13). The following table depicts an overview of the results of our experiment.

These scores indicates the degree of satisfaction of the users and provides a feedback to us in order to enhance our tool suite.

These results seem to suggest that subjects like the MDE Tool-chain as a means to speed the development of pattern-based S&D applications development by design, and in so doing, improving focus on tough tasks (e.g. implementation). However, pattern instantiation stands up as the main stumbling block for pattern-based system development adoption. More to the point, if we consider that the subjects were programmer natives (i.e. accustom to use programming language for security engineering). Specifically, users tend to overlook the three rules that govern pattern-based system development (i.e. (1) each pattern must be specified domain-application independently, (2) more than one pattern is required to fulfill one S&D property and (3) every pattern should instantiated in the target domain-development environment).

6. CONCLUSION AND FUTURE WORK

The proposed approach promotes a model-based approach coupled with a repository of models for embedded system applications, focusing on the problem of integrating non-functional properties by design to foster reuse. Currently, we are developing an MDE Tool-chain, with EMF editors and a CDO-based repository, supporting the approach. Currently the tool suite named *semcomdt* is provided as Eclipse plugins.

The approach presented here has been evaluated in the context of the TERESA project for a repository of S&D patterns and property models. For instance, a pattern designer defines patterns and store them in the repository. A system designer reuses existing patterns from the repository through instantiation mechanisms which leads to simpler and seamless designs with higher quality and costs savings.

First evidences indicate that users are satisfied with the MDE tool-chain. 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.

As future work, we plan to study the automation of the search and instantiation of models and patterns and a framework for simpler specification of constraints would be beneficial. In addition, we will study the integration of our tooling with other MDE tools, mainly those used in embedded system development. For that, we need to implement code generators able to generate a restrictive set of code complying to the domains standards.

Item	Mean	St. Dev.
1. I think the installation of the SEMCO plug-in is easy	4.10	0.48
2. I think repository populating tools are easy to use	3.80	0.54
3. I think repository access tools are easy to use	4.10	0.68
4. I think it is easy for me to develop new S&D patterns	3.50	0.36
5. I think it is easy for me to develop new properties models	3.80	0.70
6. I think S&D patterns instantiation is easy to use	3.80	0.64
7. I think properties models instantiation is easy to use	3.50	0.64
8. I would like to develop S&D patterns in the future	3.80	0.56
9. I would like to develop properties models in the future	4.10	0.68
10. I would like to install other SEMCO plugins in the future	3.50	0.54
11. I would like to exchange SEMCO in the future	3.60	0.56
12. I would like to customize some SEMCO plugins in the future	3.60	0.76
13. I would like to extend some SEMCO features in the future	3.70	0.83

Figure 12: Satisfaction Results from 1 (total disagreement) to 5 (total agreement).

7. REFERENCES

[1] 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.

[2] D. Riehle, H. Züllighoven, Understanding and using patterns in software development, TAPOS 2 (1) (1996) 3–13.

[3] D. Serrano, A. Mana, A.-D. Sotirious, Towards Precise and Certified Security Patterns, in: Proceedings of 2nd International Workshop on Secure systems methodologies using patterns (Spattern 2008), IEEE Computer Society, 2008, pp. 287–291.

[4] 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.

[5] R. B. France, B. Rumpe, Domain specific modeling, Software and System Modeling 4 (1) (2005) 1–3.

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

[7] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework 2.0, 2nd Edition, Addison-Wesley Professional, 2009.

[8] 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, IEEE, 2011, pp. 485–492.

[9] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing 1 (2004) 11–33.

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

[11] CDO Model Repository Overview. URL `http://www.eclipse.org/cdo/`

[12] Q. Omg, Meta Object Facility (MOF) 2 . 0 Query / View / Transformation Specification, Transformation (April) (2008) 1–230. URL `http://www.omg.org/spec/QVT/1.0/PDF/`