Interplay of Security&Dependability and Resource using Model-driven and Pattern-based Development

Brahim Hamid

Interplay of Security&Dependability and Resource using Model-driven and Pattern-based Development

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

***Abstract*—Several frameworks have been proposed to help designers of embedded system applications. However, we currently lack methodological tool support to take into account the interplay between security&dependability and resource properties. In this work, we propose a modeling environment which associates model-driven paradigms with security and dependability patterns to ensure that the combination of security and dependability solutions fit on the targeted hardware platform. The resulted framework will serve as a tool to estimate the resources consumed by the security and dependability solutions at early stages of design to help the designer to avoid resource conflicts at run-time. In addition, we provide an architecture for development tools to support the design and the analysis of pattern-based secure and dependable applications. Finally, we apply it in practice to a use case from railway domain with strong security and dependability requirements.**
Keywords. Security, Dependability, Resource, Pattern, Model-driven Engineering, Embedded Systems Engineering.

I. Introduction

Embedded systems share a large number of common characteristics, including real-time and physical constraints (e.g. temperature), as well as energy efficiency requirements [1]. Furthermore, many embedded systems also have assurance requirements, ranging from very strong levels involving certification (e.g. DO178 and IEC-61508 for safety-relevant embedded systems development) to lighter levels based on industry practices. Consequently, the conception and design of embedded systems is an inherently complex endeavor. In particular, non-functional requirements from Security and Dependability (S&D) are exacerbating this complexity. This is especially true for Resource Constrained Embedded Systems (RCES), as they refer to systems which have memory and/or computational processing power constraints. The generation of RCES therefore involves specific software building processes.

Security&Dependability solutions may have a vast impact to the rest of the system, for example an improved rigorous authentication mechanism may degrade performance; and vice-versa system resource restriction may potentially compromise security, for example using light cryptographic mechanisms is usually performed to boost performance but may pose security risks. Therefore the system's resource constraints and S&D cannot be considered in isolation, but they need to be studied in tandem as quality attributes. However, in many cases security and dependability engineering and system engineering are "islands", in the sense that the disciplines work independently of one another. In addition, in system engineering, S&D may be compromised in several system layers and lifecycle stages. Usually, S&D are considered when design decisions are made leading to potentially conflicting. This brings tremendous challenges during system integration and evolution. The interplay between requirements engineering and architecting has been well established [2] but we lack methods and tools to support it [3].

Model-based system engineering and Model-Driven Engineering (MDE) promote the integration of S&D issues into the complete engineering process [4]. When S&D requirements are determined, architecture and design activities are conducted using modeling-techniques and tools for higher quality and seamless development. On the other hand, the integration of S&D features using this approach requires the availability of system architecture expertise, application domain specific knowledge and S&D expertise at the same time to manage the potential consequences of design decisions on the S&D of a system and on the rest of the architecture attributes. For instance, at architectural level, security means to have a mechanism (it may be a component or integrated into a component). Hence capturing and providing this expertise by means of *S&D patterns* can enhance systems development by integrating them in the different development life-cycle stages.

In [5], we have studied pattern modeling frameworks and proposed semi-formal representation of security and dependability patterns using metamodeling techniques and provided accompanying formal specification to validate their security and dependability properties. In this

paper, we go one step further: we propose a modeling framework for the specification and analysis of secure and dependable pattern-based software and system architecture for RCES using patterns. Special emphasis will be devoted to promote the particularly challenging task of efficiently integrating security and dependability solutions, within the restricted available resources for embedded system, by design to foster reuse. At the core of the framework is a set of Domain-Specific Modeling Languages (DSML) [6] and model transformations that allow modeling a set of modeling artifacts, including pattern, resource and property models. The pattern is the first class citizen of these modeling artifacts to describe security and dependability solution. The resource will capture the computing system platform and the property will allow to govern the use of patterns and their analysis for reuse.

The rest of the paper is organized as follows. In Section II, we present a review of the most important related work. In Section III, we outline our overall approach for pattern-based engineering methodology. Section IV presents the management of S&D pattern-based architecture during the software development life-cycle with analysis activities. In Section V, we describe an model-based development framework to support the methodology. Section VI describes the usage of the defined modeling framework in the context of pattern-based RCES applications through a case study from railway domain. Finally, section VII concludes and draws future work directions.

II. Related Work

The ideas of system architecture, security and dependability modeling and analysis are not new but, to the best of our knowledge, the interplay and integration of system security&dependability and the rest of the architecture is. In this section we describe prior work in these areas and discuss their relationship to our work.

There has been a renewed interest in how to support the Twin Peaks model [7], in a wide range of aspects such as: theoretical frameworks for relating requirements and architecture, tools and techniques such as goal-oriented inference and uncertainty management, problem frames and service composition. There has also been a discussion on the similarities between the problem and solution space and the way of interpreting requirements and design decision based on the viewpoint of a stakeholder [8]. More relevant to the topic of this paper, there are also approaches for applying the Twin Peaks model in the context of security [9]. [10] presents an approach for the identification of inconsistencies of models in the context of Model-based System Engineering using pattern matching and multi-graphs based formalism.

Several proposals exist in the literature to deal with patterns for security and dependability concern. They allow to solve very general problems that appear frequently as sub-tasks in the design of systems with security and dependability requirements. These elementary tasks include secure communication, fault tolerance, etc. A framework for the development of dependable software systems based on a pattern approach is proposed in [11]. The pattern specification consists of a service-based architectural design and deployment restrictions in form of UML deployment diagrams for the different architectural services. [12] reports an empirical experience, about the adoption and eliciting S&D patterns in the Air Traffic Management (ATM) domain, and show the power of using patterns as a guidance to structure the analysis of operational aspects when they are used at the design stage. [13] 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.

In this paper, our aim is to support the interplay between S&D solutions and the resource constraints of the system using property model as an intermediate model. We will provide a set of domain-specific modeling languages and metamodels that will support software engineers in specifying the system constraints and requirements for the security and dependability and resource constraints interactions. As part of this objective we will focus on patterns to describe security and dependability solutions and on resource models to specify customized system and software architectures considering different concerns (e.g. target platforms, S&D quality attributes required by applications, etc.).

III. Approach: Contribution to the modeling of S&D applications

Security and dependability are not building blocks added to an application at the end of the life cycle. It is necessary to take into account these concerns from the requirement to the integration phases. From another perspective, in system engineering, security and dependability may be compromised in several system layers. Usually, security and dependability are considered when design decisions are made leading to potentially conflicting design requirements. The integration of security and dependability features requires the availability of a system architect, an application domain specific knowledge and a security and dependability expert at the same time to manage the potential consequences of design decisions on the security and on the dependability of a system and on the rest of the architecture. For instance, at architectural level, security means to have a mechanism (it may be a component or integrated into a component).

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). Therefore, PBSE focuses on patterns and from this viewpoint addresses two kinds of processes: the process of pattern development and the process of system development with patterns. The main concern of the first process is designing patterns for reuse and the second one is finding the adequate patterns and evaluating them with regard to the system-under-development's requirements. As well, we add a repository as a tier which acts as intermediate agent between these two processes. A repository should provide a modeling container to support modeling artifacts life-cycle associated with different methodologies.

The conceptual vision of our process model is visualized in Fig. 1. In this vision, the developer starts with the system specification fulfilling the requirements. In a traditional approach (non pattern-based approach) the developer would continue with the architecture design, module design, implementation and test. In our vision, instead of following 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 patterns from the repository and integrate them into the model of the system under development.

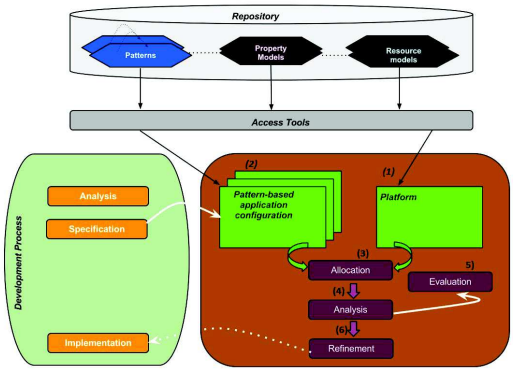

Figure 1. Pattern-based System and software Engineering

Once the repository[1] is available, the usage of this process proceeds as follow (the numbers in parentheses correspond to the numbers in Fig. 1). The developer creates a model representing the target platform importing appropriate resource models from the repository (1). For the software part, the system developer executes the search/select actions from the repository to import patterns building a pattern system configuration (2). For each of these configurations, a mapping through an allocation process is executed (3). The allocation supports the links between a pattern and its required resources from the platform model. The result of the allocation, as the platform model and a configuration for a model of a system of patterns, is then used for analyzing (4) the resource consumption of the patterns with regard to the resources defined in the platform model (e.g. memory, processing power). The result of the analysis is then delivered for evaluation (5). The developer can then use MDE techniques, such as refinement, for implementing the system using the appropriate pattern system configuration (6).

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

IV. S&D Pattern-based Architecture Modeling and Analysis

In our context, we have identified three kind of modeling artifacts: (1) *resource models* to describe platforms, (2) *S&D patterns* to describe software and system security and dependability solutions and (3) *property models* as intermediate models to serve primarily to govern the use of patterns and to evaluate their level of security as well as their resource consumption for analysis and reuse, with regard to the targeted platform. This approach aims to define an engineering discipline for S&D applications that is adapted to resource-constrained embedded systems. We now present a set of definitions and concepts that might prove useful in understanding our approach.

Adapting the definitions of [14], we propose the following:

Definition 1 (System of S&D Patterns.): We define a security and dependability system of patterns as a collection of security and dependability patterns 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 security and dependability objective.

Definition 2 (System of S&D Patterns Configuration): A system of S&D patterns configuration is a subset of a system of patterns composed of a list of S&D patterns and a list of relationships between these S&D patterns. It will be used to specify one possible structure of an application based on S&D patterns which will be deployed on a platform.

In the following, we show how we integrate a pattern from the input pattern system (i.e. from the repository) in pattern system configuration using the potential relationships between patterns. Then, we describe the analysis in terms of resource consumption to help during the pattern selection activity regarding the target platform. Mainly, we focus on the support of consistency between S&D architectural design decisions and platform resource models through patterns.

A. Pattern System Configuration Management

A system of pattern is composed of a set of patterns and their relationships (see Definition 1). Let S be a system of patterns. We denote by $P(S)$ and $R(S)$ the set of patterns and the set of the relationships of the system of patterns S, respectively. Each relationship is associated with a relationship type [15], denoted by $referenceKind$, and each pattern is associated with a list of its relationships, denoted by $references$. Let Sc and Si denote the complete system of patterns and the initial configuration of the system of patterns, respectively. To define a set of system of patterns configurations, denoted by Sg, we propose in Listing 1 a simple algorithm called $configurationGen$.

```
1  Algorithm configurationGen
2      Input: Sc, Si, referenceKind.
3    Output: {Sg}.
4   for each Pattern in the P(Si)  do
5    for each Reference in Pattern.references do
6      if Reference.referenceKind  = referenceKind
7        duplicateModel(Sg, Si)
8        replacePattern(Sg, Sc, Pattern,
             referencedPattern)
9        save()
10       endif
11   endfor
12  endfor
```

Listing 1. Configuration generation algorithm

This algorithm takes as input Sc and Si and a referenceKind from $R(Sc)$ to find further configurations which are equal to Si with respect to the relationship (i.e. referenceKind). The base of the system of patterns (Si) is parsed and for each pattern in this system equal patterns (with respect to the referenceKind) are looked up in the complete system of patterns (lines 3 and 4). If a pattern is found, a new configuration ($newConfig$) is generated from the base system (line 7) and the pattern ($Pattern$) from Si is replaced by the equal pattern ($referencedPattern$) from Sc. If multiple patterns are equal to a pattern in the base system, multiple configurations (one for each equal pattern) are generated. This algorithm allows to generate equal (e.g. similar, alternative) pattern system configurations and, later on, to analyze these different configurations.

B. Pattern-based System Model Analysis

Once the platform was specified using resource models, the analysis activity will be used to calculate resource consumption values and to incorporate these information into system models. For that, we provide in Listing 2 a simple algorithm to calculate the resource usage of a pattern system configuration (Sg) for a platform (R). This algorithm parses the used patterns for their consumption of different resources defined in the platform model (e.g. memory, processing power) and adds these up. The result is then injected into the platform model.

```
1  Algorithm CalculateResourceUsage
2     Input: Sg as a system of pattern configuration
          model
3     InputOutput: R as a platform model
4   for each Property pr in R
5      for each Pattern p in the P(Sg)  do
6        for each P.Property do
7          if (P.Property.category=pr.category) then
8          pr.value:=sum(pr.value,P.Property.value)
9        endfor
10     endfor
11  endfor
```

Listing 2. Resource consumption computing algorithm

For each identified property, the algorithm loops over all the patterns composing the system of patterns configuration (Sg). For each pattern (line 5), all its properties' categories (line 6) are compared to the category of the R resource property, and if matching (line 7) the values are summed up (line 8). The result is then stored as the value of the value of the platform resource property.

V. Model-Based Development (MBD)

In this section we describe an MDE framework to support the previous approach. We use metamodeling and model transformation techniques for the specification and analysis of secure and dependable system and software architecture. However, the approach does not prescribe a fixed set of metamodels and model transformations to be used. As mentioned earlier, the approach only focuses on security and dependability requirements that directly influence the available platform resources.

A. Artifacts Modeling Languages

Here, we provide a description of a set of modeling languages for the specification of the identified set of modeling artifacts: S&D patterns, property and resource models. As we shall see, S&D and resource models are used as model libraries to define the S&D and resource properties of the pattern. Therefore, a pattern can be stored in a repository and can be loaded according to the desired S&D and resource properties.

1) A Metamodel for Non-Functional Properties (GPRM).: The Generic PRoperty Metamodel (GPRM) is a metamodel defining a new formalism for describing property libraries including units, types and property categories. The following paragraph details the meanings of the principal classes of the *GPRM Metamodel*, which is depicted with Ecore notations in Fig. 2.

- *GprmProperty.* A property is a basic attribute shared by all members of an artifact (pattern, resource, etc). As we shall see, it will be used to define pattern and resource properties (see SERM and SEPM metamodels).

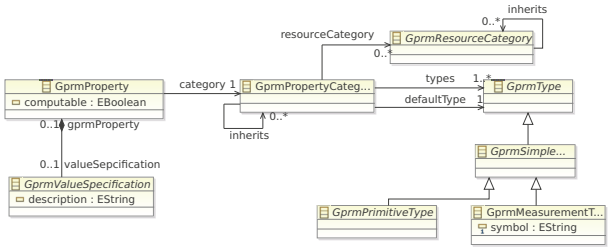

Figure 2. The (simplified) GPRM Metamodel

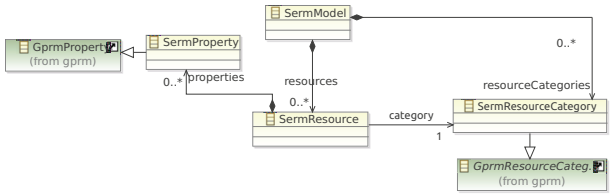

Figure 3. The (simplified) SERM Metamodel

The property is defined by its category, its type and its value.
Example. CPU execution time for encryption and energy consumption for encryption are two properties (resource-related) of the pattern SecureCommSSL.

- *GprmPropertyCategory.* A property category is a classification for properties. Its role is to group all the properties sharing common characteristics. These characteristics may depend on the user or application domain viewpoint. A category supports a set of types that define the nature of the property, it can also be defined based on other categories by specialization. A category is defined with at least one default type.
  Example: CPUTime is a category of resource properties and *Authenticity* is a category of S&D properties.
- *GprmResourceCategory.* A resource category is a classification for resources. Its role is to group all the resources sharing common characteristics. These characteristics may depend on the user or application domain viewpoint. The main categories are those related to computing, data storage and to energy consumption. However other categories may be defined such as peripheral, sensor, actuator, etc. A category may also be built based on existing categories by specialization.

2) A Metamodel for Resource (SERM).: In the development context defined in our approach, embedded platforms are seen as a composition of (hardware) resources. Resources are pieces of the platform and can be combined and linked together without having an external observable state. The System and Software Engineering Resource Metamodel (SERM) is a metamodel for describing resources and their properties, as described with Ecore notations in Fig. 3. The meanings of the principal classes are more detailed in the following paragraph.

- *SermResource.* Is a modeling artifact which represents a piece of the platform. It defines a set of parameters that will be later used by the framework to automate the analysis of applications based on S&D patterns.
  Example. IntelAtom Z530 and *DDR2 RAM* are two hardware resources for computing and data storage, respectively.
- *SermResourceCategory.* It is an *GprmResourceCategory* denoting a classification of a resource. *Example.* As computing resource categories we can define *CPU*, *FPGA*, *DSP*, etc. Therefore, the *IntelAtom Z530* belongs on *CPU* resource category and the *DDR2 RAM* belongs on *RAM* resource category.
- *SermProperty.* It is an *GprmProperty* denoting a particular characteristic of a resource.

3) A Metamodel for S&D Patterns (SEPM).: The System and Software Engineering Pattern Metamodel (SEPM) is a metamodel 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 paragraph details the principal concepts of the SEPM metamodel to specify an S&D pattern, as described with Ecore notations in Fig. 4.

- *SepmPattern.* This block represents a security pattern as a subsystem describing a solution for a security particular recurring design problem that arises in specific design context. A *SepmPattern* defines its behavior in terms of provided and required interfaces.
- *Interface.* A *SepmPattern* interacts with its environment with *Interfaces*. We consider two kinds of interface:
  - *SepmExternalInterface.* Allows implementing interaction with regard to the integration of a pattern into an application model or to compose patterns.
  - *SepmTechnicalInterface.* Allows implementing interaction with security primitives and protocols, such as encryption, and specialization for specific underlying software and/or hardware platforms, mainly during the deployment activity.
- *SepmProperty.* Is an *GprmProperty* denoting a particular characteristic of a pattern related to the concern it is dealing with and dedicated to capture its intent in a certain way. For instance, security and dependability properties.

Example. We illustrate the usage of the SEPM for specifying a pattern with the example of secure communication pattern based on SSL[2] mechanism. Here, we specify an S&D property: "authenticity of sender and receiver". To type the category of this property we use a category from the earlier defined in the S&D category library: *Authenticity*. Moreover, we identify some resource properties, such as "CPU resource time for encryption" and "CPU resource time

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

for authentication" that belong to category *CPUTime*, and "extra energy cost for encryption" and "extra energy cost for authentication" that belong to category *PowerConsumption*.

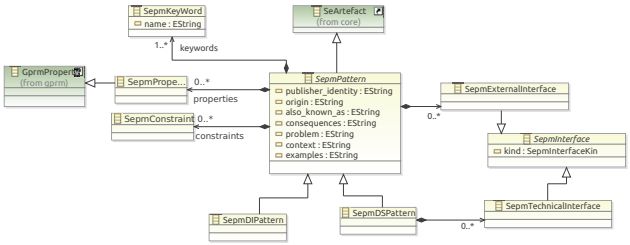

Figure 4. The (simplified) SEPM Metamodel

4) A Metamodel of system of patterns.: According to Definition 1, an S&D pattern system is a modeling artifact system where its constituent parts are S&D patterns and their potential relationships. In the following we detail the meanings of the principal classes of the *pattern system metamodel*, which is depicted with Ecore notations in Fig. 5.

- *SepmSystemOfPatterns.* A system of S&D patterns describes the relationships and the linkage among individual patterns (*SeReference*). Thus dependencies between specific problems can be considered in a comprehensive way.
- *SeReferenceKind.* We use an enumeration to define a set of types of links between S&D patterns as an extension of the relationships classification proposed in [15].
  - *refines.* It is used to represent the refinement relationship between two patterns.
  - *specializes.* It is used to represent the specialization relationship (detail).
  - *uses.* It is used to represent the functional dependency relationship between two patterns.
  - *isSimilar.* It allows to link two patterns that perform the same functionality. This link is often used to link software patterns to their equivalent hardware patterns.
  - *isAnAlternative.* It allows to link tow patterns that solve the same problem, but propose different solutions.

B. System of Patterns Configuration Analysis

For the analysis, we propose to use model transformations techniques to calculate resource consumption values and to incorporate the results into system models. According to the describable artifact, we introduce a new kind of properties to be described as a computable characteristic using the attribute *computable* (see Figure 2). Hence, the value of the property could be defined either by the designer during the design activity or by calculation during the analysis activity. In this case, the corresponding instance of *GprmProperty* (GPRM) has the *computable* attribute set to "true".

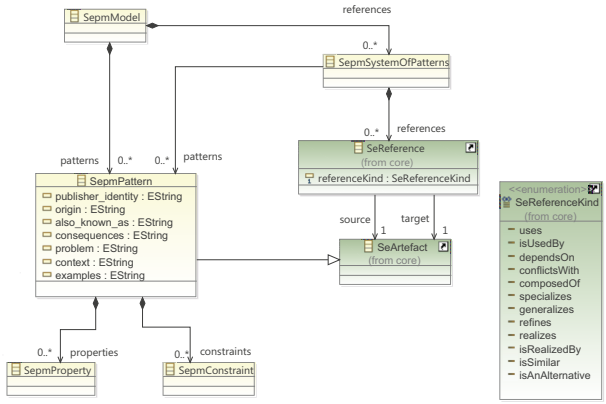

Figure 5. A metamodel of a system of patterns

The overview of the framework's transformations is shown in Fig. 6. The concepts of the transformations are expressed (i.e. Transformation Specification) using the Source Metamodel(s) and the Target Metamodel(s). Once specified, transformation is run (i.e. Transformation execution) taking as input the Source Model(s) and producing as output the Target model(s).

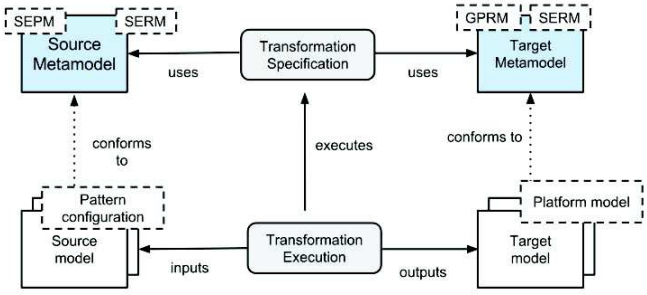

Figure 6. Overview of the framework's transformations

C. Design and Analysis Tools

Using the proposed metamodels and the Eclipse Modeling Framework (EMF) [16], ongoing experimental work is done with SEMCOMDT [3](SEMCO Model Development Tools, IRIT's editor and platform plugins) as a Model-Driven Engineering tool-chain supporting the proposed metamodels. All used metamodels are specified using EMF. For the description of the model transformations, the QVT Operational language[4] is used. We build a set of software tools, for instance for design, for populating the repository and for retrieval and transform from the repository. Moreover, we provide tools to support the

[3]http://www.semcomdt.org
[4]http://www.omg.org/spec/QVT/

management of systems of patterns, the generation of configurations and the transformations for analysis.

The resource designer called *Matho* includes features supporting the design of resource category libraries, t design of platform resources, deposit to and retrie from the repository through access tools. Although t tool allows to mix resource categories and resources the same model (conforming to the definition of SER metamodel), we recommend to separate these artifa in two different models. The first is used to define t libraries of resource categories; and the second is us to describe the platform resources by importing a using the previous resource category libraries to type t resources. The pattern designer called *Arabion* is us to populate the repository with patterns conforming SEPM metamodel, to model the system of patterns a to derive configurations from systems of patterns.

Using the algorithm proposed in IV-A, we generate a set of configurations that will be simulated with the platform in order to select the ones that give the best resources consumption trade-off. To traverse the model of the platform, the pattern system configuration and to perform calculations of resource consumption, such as depicted in Listing 2, we developed a M2M transformation using the Eclipse implementation of QVTO.

VI. A Case Study

We demonstrate the applicability of our proposed framework through the *Safe4Rail* demonstrator [17] which is a simplified version of a real ETCS (European Train Control System) signaling, control and train protection system. The main functionality of this demonstrator is to supervise that traveled speed and distance do not exceed authorized maximum values provided by the railway infrastructure. In the following sub-sections, we focus on the European Vital Computer (EVC) subsystem' platform that executes the safety application. For the sake of simplicity, many functionalities of this case study have been omitted. As we will see, the architecture will be incomplete, since the approach only focuses on the description of the system platform that directly influence the security and dependability requirements. The hardware part of this subsystem is composed of a carried board on which is installed a conga-CA board with a microprocessor (Intel Atom Z530), a RAM (DDR2 RAM) and a set of interfaces for the connections. The conga-CA is accompanied with an additional programmable resource calculation unit (FPGA-Spartran).

A. Identification of Patterns

The architects analyze system safety and security requirements and mentally identify possible architectures and safety solutions to be used. Fig. 7 shows how these patterns are interconnected to form the whole system of patterns.

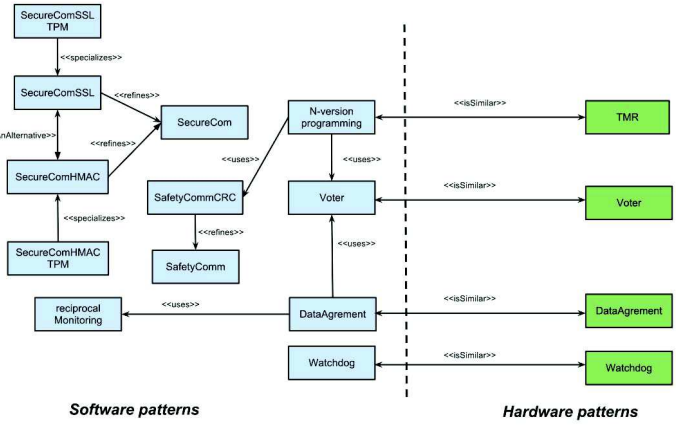

Figure 7. Architecture of complete system of patterns

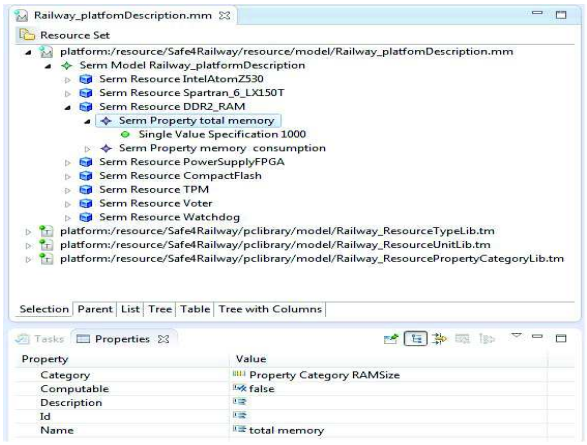

Figure 8. Safe4Rail platform description

B. Modeling of Safe4Rail

First, we model the platform resources using the predefined libraries of resource categories. We have used *Matho* which provides a tree editor for resources modeling, as shown in Fig. 8. These resources are tagged and configured using the *resource category libraries* and the *property category libraries*, so that the platform model can be built subsequently as a set of configured resources. Second, we model the application based on the selected S&D patterns. We have used *Arabion* following modeling steps as explained in Section V-C. This means that we have started by modeling the whole pattern system including all the patterns and the relationships between them as shown in Fig. 7. Then, we use *Arabion* again to build the base configuration. In our case, we

			Scenario 1		*Scenario 2*		*Scenario 3*	
Resources	Resource property category	Max value	Usage	%	Usage	%	Usage	%
IntelAtom Z530	CPUTime	3,600	379.00	10.53%	339	9.42%	533	14.81%
Spartran 6 LX150T	FPGATime	2,000	0.00	0.00%	0	0.00%	0	0.00%
DDR2_RAM	ramsize	1,000	522.00	52.20%	398	39.80%	398	39.80%
PowerSupplyFPGA	powerConsumption	50	44.00	88.00%	36	72.00%	50	100.00%
CompactFlash	memorysize	32	2.00	6.25%	4	12.50%	0	0.00%
TPM	TPMsize	2	0.00	0.00%	0	0.00%	1	50.00%
	TPMtime	1,000	0.00	0.00%	0	0.00%	50	5.00%
Voter	Votertime	500	50.00	10.00%	0	0.00%	0	0.00%
WatchdogHW	WatchdogTime	200	0.00	0.00%	0	0.00%	0	0.00%

Table I
Analysis of the three scenarios

have chosen an initial configuration composed of *N-version programming, VoterSW, DataAgrement, Watchdog, SafetyCommCRC, SecureCommSSL* and *reciprocal monitoring* patterns. Finally, using the configuration generation algorithm introduced in Section IV-A, we get a set of pattern system configurations. Next, we will see how some of these configurations will be used for the resource consumption analysis.

C. Analysis of Safe4Rail Application

To describe the conduct of analysis, we have selected three configurations:

- Scenario 1: "isSimilar" configuration is obtained by replacing *sofware Voter* pattern by *hardware Voter* pattern in the "Base" configuration.
- Scenario 2: "isAlternative" configuration is obtained by replacing *SecureComSSL* pattern by *SecureComHMAC* pattern in the "Base" configuration.
- Scenario 3: "Specializes" configuration is obtained by replacing *SecureComSSL* pattern by *SecureComSSLTPM* pattern.

Both the system of patterns and the platform are modeled as shown previously. To calculate the whole resource consumption, we execute the M2M transformation engine, using the configuration model of this scenario and the platform model as inputs, to produce a new platform model annotated with the calculated values of the resource consumption. Table I shows a comparative view of the resource consumption of the three pattern system configurations. Of course, since the system architecture contains only elements necessary for the description of the platform, the architecture will be incomplete. However, the resulted model of the system, including platform description, system of patterns configurations and the analysis, will allow the architect to experiment and to get familiar with different possibilities and options before proceed to the next steps. Also, the resource consumption results can be used to analyze which security and dependability requirements should be implemented and which should be candidates for modification or for elimination.

VII. Conclusion and Future Work

The intended role of pattern-use is to ease, systematize and standardize the approach to the construction of software-based systems. However, the problem consists in identifying them explicitly and then selecting them for reuse. In Resource Constrained Embedded Systems (RCES) development, this means that a careful balance and trade-off analysis among Security and Dependability (S&D) requirements, patterns and available platform resources is necessary. In doing so, we propose to use metamodeling and model transformation techniques for the specification of pattern configuration and to calculate resource consumption values and to incorporate this information into system models. This allows: (1) To define a new paradigm for design and implementation based on the interplay between the S&D and resource properties; (2) To increasing S&D level of RECS products by making S&D a built-in feature while enabling a consistent architecture of a system taken into account resource constraints. Furthermore, we walk through a prototype as a set of EMF editors and a set of model transformation engines supporting the approach using Operational QVT specification. An illustration by means of a case study from railway domain is provided.

As stated earlier, the pattern-based security and dependability methodology seems to be a very interesting combination of two flavors of software and hardware design which has not been addressed sufficiently in literature. The next step of this work consists in defining a pattern-based security engineering methodology. It aims at providing the correct-by-construction integration of a design pattern into an application while offering a certain degree of liberty to the designer using it. In order to be able to validate the integration, we must have a formal specification of the pattern [18], i.e., its properties, constraints and related validation artifacts [19], as input to the pattern-based development process. The architecture description needs to be extended with other type of elements, e.g., software components, connectors and deployments units. This types of elements can be used, for instance, to describe architectural security and dependability patterns [20]. Concurrently, more sophisticated techniques to derive artifacts relationships can

be implemented, possibly using different domains, to reduce the complexity to design pattern systems. Furthermore, the correctness of the model transformations have to be determined. Another objective for the near future is to provide automated tool support for pattern-based development, preferably based on a widely known and accepted model-based approach in industry such as UML [21]. For that, we plan to investigate the possibility to transform our design artifacts into UML and their corresponding validation artifacts into OCL [22], which would make these patterns compatible with collections such as the one in [23].

References

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

[2] B. Nuseibeh, "Weaving together requirements and architectures," *IEEE Computer*, vol. 34, no. 3, pp. 115–117, 2001.

[3] M. Galster, M. Mirakhorli, J. Cleland-Huang, J. Burge, X. Franch, R. Roshandel, and P. Avgeriou, "Views on software engineering from the twin peaks of requirements and architecture," *SIGSOFT Softw. Eng. Notes*, vol. 38, no. 5, pp. 40–42, 2013.

[4] B. Hamid, J. Geisel, A. Ziani, J. Bruel, and J. Perez, "Model-Driven Engineering for Trusted Embedded Systems Based on Security and Dependability Patterns," in *SDL Forum*, pp. 72–90, 2013.

[5] B. Hamid, S.Gurgens, C. Jouvray, and N. Desnos, "Enforcing S&D Pattern Design in RCES with Modeling and Formal Approaches," in *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*, vol. 6981, pp. 319–333, Springer, octobre 2011.

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

[7] P. Avgeriou, J. Grundy, J. G. Hall, P. Lago, and I. Mistrík, eds., *Relating Software Requirements and Architectures*. Springer, 2011.

[8] R. de Boer and H. V. Vliet, "On the similarity between requirements and architecture," *Journal of Systems and Software*, vol. 82, no. 3, pp. 544–550, 2009.

[9] T. Heyman, K. Yskout, R. Scandariato, H. Schmidt, and Y. Yu, "The security twin peaks," in *Proceedings of the International Symposium on Engineering Secure Software and Systems (ESSoS)*, vol. LNCS 6542 of *Lecture Notes in Computer Science*, pp. 167–180, Springer, 2011.

[10] S. J. Herzig, A. Qamar, and C. J. Paredis, "An approach to identifying inconsistencies in model-based systems engineering," *Procedia Computer Science*, vol. 28, no. 0, pp. 354 – 362, 2014.

[11] M. Tichy, D. Schilling, and H. Giese, "Design of self-managing dependable systems with UML and fault tolerance patterns," in *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, WOSS '04, pp. 105–109, ACM, 2004.

[12] V. D. Giacomo, M. Felici, V. Meduri, D. Presenza, C. Riccucci, and A. Tedeschi, "Using Security and Dependability Patterns for Reaction Processes," in *Proceedings of the 2008 19th International Conference on Database and Expert Systems Application*, pp. 315–319, IEEE Computer Society, 2008.

[13] E. B. Fernandez, N. Yoshioka, H. Washizaki, J. Jrjens, M. VanHilst, and G. Pernul, "Software Engineering for Secure Systems: Industrial and Research Perspectives. In H. Mouratidis, editor," *IGI Global*, pp. 16–31, 2010.

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

[15] J. Noble, "Classifying relationships between object-oriented design patterns," in *Software Engineering Conference, 1998. Proceedings. 1998 Australian*, pp. 98–107, 1998.

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

[17] TERESA, "Specification of Platform. Deliverable D6.1 – TERESA/WP6/D6.1, IST Project IST-248410," January 2013.

[18] B. Hamid and C. Percebois, "A modeling and formal approach for the precise specification of security patterns," in *Engineering Secure Software and Systems - 6th International Symposium, ESSoS 2014*, vol. 8364 of *Lecture Notes in Computer Science*, pp. 95–112, Springer, 2014.

[19] S. Gürgens, P. Ochsenschläger, and C. Rudolph, "On a formal framework for security properties," *International Computer Standards & Interface Journal (CSI), Special issue on formal methods, techniques and tools for secure and reliable applications*, vol. 27, no. 5, pp. 457–466, 2005.

[20] U. Zdun and P. Avgeriou, "A catalog of architectural primitives for modeling architectural patterns," *Information & Software Technology*, vol. 50, no. 9-10, pp. 1003–1034, 2008.

[21] OMG, "Unified Modeling Language, v2.4.1," 2012.

[22] OMG, "OCL 2.2 Specification," February 2010.

[23] E.B.Fernandez, *Security patterns in practice: Building secure architectures using software patterns*. Wiley Series on Software Design Patterns, 2013.