Modeling Security and Dependability Patterns in Resource Constrained Embedded Systems

Brahim Hamid[1], Nicolas Desnos[1], David Gonzalez[2] and Manuel Blanco[2]

[1] IRIT, University of Toulouse
118 Route de Narbonne, 31062 Toulouse Cedex 9
France
brahim.hamid,nicolas.desnos@irit.fr
[2] IKERLAN-IK4
Mondragon, Spain
DGonzalez,MFBlanco@ikerlan.es

Abstract. The requirement for higher reliability and availability of systems is continuously increasing even in domains not traditionally strongly involved in such issues. Particularly, in Resource Constrained Embedded Systems (RCES) solutions are expected to be efficient, flexible, reusable on rapidly evolving hardware and of course at low cost. Model driven approaches can be very helpful for this purpose. In our work, we propose a study associating model driven technology and patterns development to build Security and Dependability (S&D) RECS applications.
The contribution of this paper is focused on design technique to assist developers of S&D patterns. We use meta-modeling techniques to capture the structure of S&D patterns at even greater level of abstraction. Then, modeling techniques are used to define pattern and to specify them for several domains. Hence, models are used as first class citzen all along the development process. This is yield an homogeneous way to catalog them in the form of repository of models and to integrate them in a MDE process to build trusted applications in RCES for several domains. As a proof of concept, we examine a dependability pattern from industrial control system domain: a Majority Voter.
Key words: Resource Constrained Embedded Systems, Security, Dependability, Patterns, Meta-model, Model Driven Engineering.

1 Introduction

Resource Constrained Embedded Systems (RCES) refers to systems which have memory and/or computational processing power constraints. RCES are becoming increasingly complex and have various communication interfaces. Therefore, they have to be seen in the context of bigger systems or complete infrastructures. Consequently, their non functional requirements such as security and dependability (S&D) [18] become more important as well as more difficult to achieve. The integration of S&D requires the availability of both application expertise and S&D expertise at the same time. Most organizations developing RCES have limited S&D expertise.

The concept of security and dependability patterns [23, 11, 24, 6, 22] as a well-understood solution to a recurring information security and dependability problem was

introduced to support the system engineer in selecting appropriate security or dependability solutions. However, most security patterns are expressed in a textual form, as informal indications on how to solve some (usually organizational) security problems. Some of them use more precise representations based on UML diagrams, but these patterns do not include sufficient semantic descriptions in order to automate their processing and to extend their use. Furthermore, there is no guarantee of the correct application of a pattern because the description does not consider the effects of interactions, adaptation and combination. This makes them not appropriate for automated processing within a tool-supported development process. Finally, because this type of patterns is not designed to be integrated into the user systems but to be implemented manually, the problem of incorrect implementation (the most important source of security problems) remains unsolved.

In RCES systems, solutions are expected to be efficient, flexible, reusable on rapidly evolving hardware and of course at low cost [20]. Solutions are usually concrete technologies for a specific domain (avionics, automotive, transports and energy). Model-Driven Engineering (MDE) provides a very useful contribution for the design of trusted systems, since it bridges the gap between design issues and implementation concerns. It helps the designer to specify in a separate way non-functional requirements such as security and/or dependability issues at an even greater level that are very important to guide the implementation process. The TERESA project[3] is aimed at defining, demonstrating and validating an engineering discipline for trust that is adapted to resource constrained embedded systems. We propose a study associating model-driven technology and patterns development to deal with the design of trusted applications.

While most of works about patterns do not propose flexible and generic techniques to encode them, this paper proposes a model-based patterns for security and dependability. The formalization is based on common concepts related to a number of sectors in RCES (automotive, home control, industry control, metering, ..). The goal here is not merely to explore patterns properties, but to identify special artifacts that can be used for specifying S&D patterns in RCES. That is, we seek a language that aims at structuring them in order to make easy their use in a software building process. To achieve this purpose, we build on three main currents in S&D patterns research: (i) To give an understanding level for each actor using these patterns, for example security expert and domain expert, (ii) To find a structured way to express security and dependability issues in a simple way, (iii) To express dependencies between patterns.

The rest of this paper is organized as follows. Section 2 provides some reminders about tools used in our contribution. In Section 3, we introduce our approach through a short description. Then, Section 4 details our modeling framework to encode S&D patterns in the context of RCES. As a proof of concept, we examine in Section 5 a notable example with dependability requirements: Majority Voter. In Section 6, we briefly review most related work. Finally, Section 7 concludes this paper with a short discussion about future works.

[3] This work has been performed in the context of the FP7 TERESA project (http://www.teresa-project.org).

2 Background

The supporting research of Security and dependability includes system architecture, design techniques, validation, modeling, software reliability and real time processing. Before we embark on discussing any aspect of security and dependability in RCES, we have to outline different tools used in our proposed approach including RCES, S&D patterns and MDE.

2.1 Resource Constrained Embedded Systems

An embedded system [25, 13, 15] is a system that is composed of two mainly parts, software and hardware, and which evolves in a real world environment. Embedded systems concern several domains like: aerospace, military, communication (mobile, gps,...), medical, automotive, avionic, home control, the industry control, the metering sector. Embedded systems are not classical software, which can be built by using classical paradigms. Indeed, software and hardware constraints must be taken in account together.

Resource constrained embedded systems (RCES) refers to systems which have memory and/or computational processing power constraints. They can be found literally everywhere, in many application sectors such as automotive, aerospace, and home control. They are in many types of devices, like sensors, automotive electronic control units, intelligent switches, and home appliances such as washing machines and meters. In addition, they have different form factors, e.g. standalone systems, peripheral subsystems, and main computing systems. Many RCES 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.

2.2 Patterns and S&D Patterns

The concept of pattern was first introduced by Alexander [2]. A pattern deals with a specific, recurring problem in the design or implementation of a software system. It captures expertise in the form of reusable architecture design themes and styles, which can be reused even when algorithms, components implementations, or frameworks cannot. We adopt from Buschmann [3] the following definition: *A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution*. That is, patterns have been considered as a suited solution to enhance the construction of software with support of functional and non functional properties.

Today, design patterns are considered as fundamental technique to build software by capitalizing knowledge to solve occurring problems (in many specific domains). Design patterns are medium-scale patterns comparing to architectural patterns but they are at a higher level than the programming language. The application of a design pattern has no effect on the fundamental structure of a software system, but may have a strong influence on the architecture of a subsystem (components).

With regard to security and dependability aspects, Yoder and Barcalow [23] were the first to work on security pattern documentation. However the typical structure of a security pattern is presented in [20].

2.3 Model Driven Engineering

Model Driven Engineering (MDE) is a form of generative engineering [19], in which all or a part of an application is generated from models. It looks promising since it offers tools to deal with the development of complex systems improving their quality and reducing their development cycles. The development is based on model approaches, meta-modeling, development process and execution platforms.

As presented in [9], MDE may be considered form tow points of view: methodologists and developers. From methodologists an MDE process should define levels of abstraction, the modeling notations, the abstract syntax, how refinements are performed, how can a model be verified against the upper level model and how can it be validated. Developers consider the application of an MDE process as a models driven refinement steps.

2.4 Role in S&D for RCES

The integration of S&D in RCES requires the availability of both application expertise and S&D expertise at the same time. In fact, S&D could also require both specific security expertise and specific dependability expertise. Most organizations developing RCES have limited S&D expertise. In software engineering, patterns are considered as an efficient tool to reuse specific knowledge. For security & dependability we can encapsulate some experience in the design of such systems through the definition of specific design patterns. For instance, S&D patterns are well suited to be used in embedded real time systems. Then, the implementation may be achieved using model based engineering tools devoted to this field.

In this work, we focus on patterns specification providing two levels of abstraction: generic and specific. For S&D patterns, we can encapsulate some experience in their design through the definition of a set of artifacts and levels of abstraction. It follows, we use modeling tools such as UML, its profiles and MDE to specialize such patterns to specific domains.

3 Our approach in a Nutshell

There are several representations of patterns. For example textual descriptions [10], diagrams with notations such as UML object, most often accompanied by examples of code to complete the description. When restricted to use pattern for documentation, or when pattern are simply used for modeling, these representations are quit sufficient. However, these representations are not useless in the development of systems.

The key ideas of the new approach presented here are: To capture appropriate characteristics of security and dependability patterns in RCES, to utilize several views, to limit the information required to build patterns, and to limit the number of interactions

with non specialized users to choose a pattern. Applying these ideas, we obtain a new engineering process to develop S&D patterns with the desired complexity of use.

The main difficulty to overcome in the implementation of S&D patterns in RCES is how to avoid the cost of building a pattern for each S&D properties and/or for each domain. One way to obtain high level of abstraction is to make use of meta-modeling techniques. Informally, a pattern has several views with regard to the considered level of abstraction. This decomposition and separation of uses illuminates how to create, to specialize and to store patterns. This implies that a pattern is created at high level abstraction and then it will be transformed into more specific one. This can be accomplished by combining meta-modeling and model driven engineering techniques such that specific artifacts are derived from more generic ones. The main idea of our proposal is to use three levels of abstraction. Then, each level (see Fig. 1) can be created form artifacts of the higher level. In our concerns, we propose the following levels of abstraction:

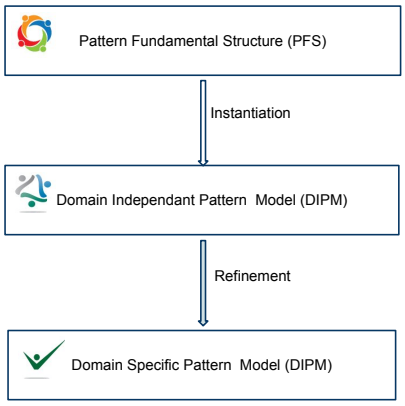

Fig. 1. An overview of the proposed formalization

The key ideas of the approach presented here are: To capture appropriate characteristics of security and dependability patterns in RCES and to utilize several views. Applying these ideas, we obtain a new engineering process to develop S&D patterns with the desired complexity of use. The technique, as shown in Fig. 1, is based on three levels of abstraction:

1. *Pattern Fundamental Structure (PFS)*: At the pattern fundamental structure level, a meta-model defines a flexible structure of patterns with S&D and RCES properties;
2. *Domain Independent Patten Model (DIPM)*: Then, it is possible to specify and develop patterns by a model. First, the pattern model is domain independent and;
3. *Domain Specific Pattern Model (DSPM)*: Finally, it is refined for adding domain specific informations.

Each level is discussed in the next sub-sections. We sketch a few notable example next. For clarity's sake, we use UML notations to describe such levels.

4 S&D Patterns Development Life Cycle

The goal is to construct a common representations of S&D patterns for several domains in RCES. Now we explain in depth the three layers and we sketch a few notable example next.

4.1 Fundamental Structure of S&D Patterns (PFS)

This section is about the representation of S&D patterns at a high level of abstraction which we shall call *PFS*. A natural question which arises is whether we can use meta-modeling technology to improve patterns representation. A naive way to model patterns is to associate concepts with each of the fields used by [10] and [20]. Note, however, that this way is not useless to capture specific characteristics of S&D patterns.

The Pattern Fundamental Structure (PFS) is a meta-model which defines a new formalism for defining S&D patterns. The PFS is presented in the right side of Fig. 2. The originality of this approach is to consider patterns as building blocks that expose services (via interfaces) and manage S&D and RCES properties (via features) yielding a way to to capture meta-information related to patterns and their context of use. As we shall see in the left side of Fig. 2, we based our representation on a component vision.

The follow paragraphs detail the principals classes of our meta-model–all described in form of UML notations:

- *FPattern*. This block represents a modular part of a system that encapsulates a solution to a recurrent problem. A *FPattern* defines its behavior in terms of provided and required interfaces. Such a *FPattern* serves as a type whose conformance is defined by these provided and required interfaces. One *FPattern* may therefore be substituted by another only if the two are type conformant. Larger pieces of a system's functionality may be assembled by reusing patterns as parts in an encompassing pattern or assembly of patterns, and wiring together their required and provided interfaces. A *FPattern* is modeled throughout the development life cycle and successively refined into deployment and run-time. A *FPattern* may be manifest by one or more artifacts, and in turn, that artifact may be deployed to its execution environment. A deployment specification may define values that parameterize the pattern's execution.
- *FIdentity*. This is the identity card of the pattern. This artifact is based on the GoF [10] information.
- *FFeature*. This information allows one to classify and to configure pattern's parameters (e.g. S&D parameters). Another issue is expressing the services provided by the *FPattern*. It can be implemented as: *Properties* to classify and to configure some parameters, for example S&D parameters, and to express the services provided by the pattern. Properties can be used by configuration tools to preset configuration values of a *FPattern*.

– *FInterface*. A *FPattern* possesses provided and required interfaces. A provided interface is implemented by the *FPattern* and highlights the services exposed to interact with its environment. A required interface corresponds to services needed by the pattern.
– *FInternalStructure*. It describes the implementation of the *FPattern*. Thus the InternalStructure can be considered as a white box which exposes the details of the *FPattern*.

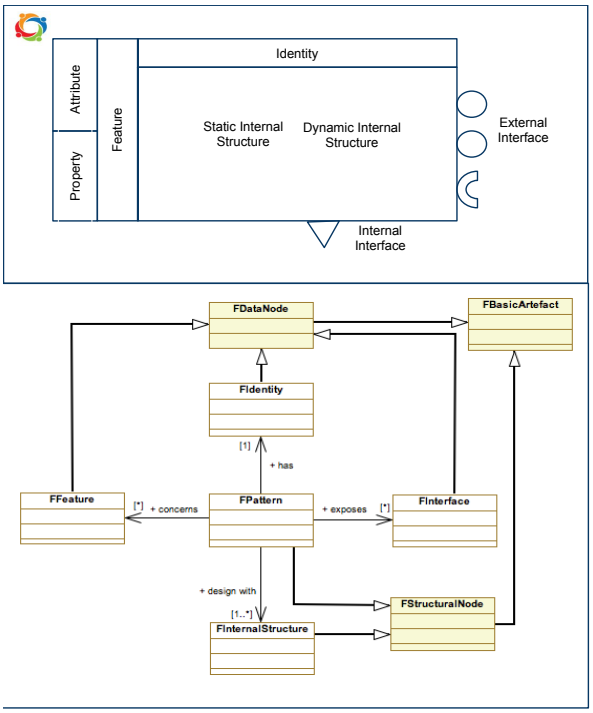

Fig. 2. Pattern Fundamental Structure

4.2 Domain Independent Pattern Model

This level describes artifacts at middle level of abstraction. This is an instance of the PFS. As a side remark, at this level a pattern contains generic information to specialize for a specific domain. Figure 3 depicts the representation of a S&D pattern at this level. As we shall see, we introduce new concepts through instantiation of existing concepts of the PFS meta-model in order to cover most existing S&D patterns in RCES applications.

We outline artifacts related to pattern domain independent level. That is,

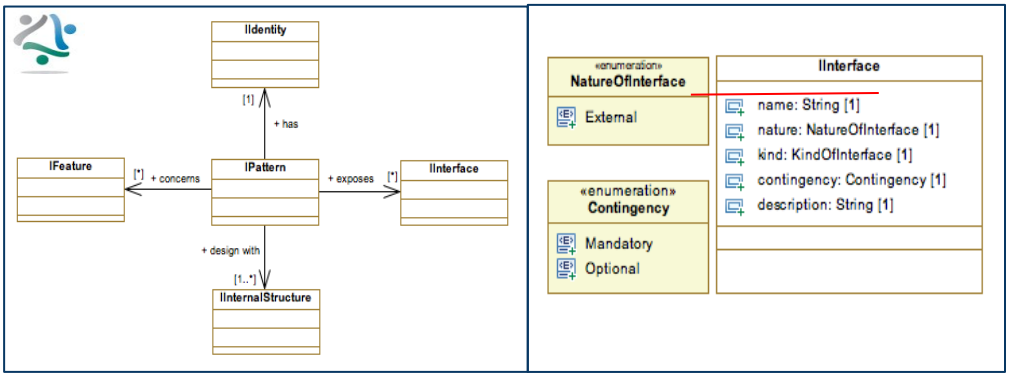

Fig. 3. S&D patterns at domain independent level

- *IPattern*. This is the representation of a FPattern at the model level. It corresponds to an instance of a *FPattern*.
- *IProperty*. One can define several kinds of properties. For instance, in the context of RCES we propose to address the following standards non-functional properties:
  - *Security:* AccessControl, Integrity, Authenticity, Confidentiality, ...;
  - *Dependability:* Availability, Reliability, Maintenability, ...;
  - *RCES:* timeDelay, RamSize, ...;
- *IInterfaces*. It is necessary to instantiate the interfaces of a FPattern. For instance, we consider *External Interfaces* nature that allow implementing interaction with regard to:
  - *integrate* a *IPattern* to an application model. These interfaces are realized by the *IPattern*.
  - *compose IPatterns* together.
- *IInternalStructure*. A *Domain Independent Pattern* has an internal structure that describes its implementation. This one is composed of static elements in order to form the structure and a set of behavior properties in order to describe its behavior.

4.3 Domain Specific Pattern Model

The objective of the specific design level, as shown in Figure 4 is to specify the S&D patterns for a specific application domain. This level offers artifacts at down level of abstraction with more precise *information* and *constraints* about the target domain.

We outline artifacts related to pattern domain specific level. That is,

- *SPattern*. It refines the *IPattern*. Several *SPatterns* can be built from a same *Domain Independent Pattern*.
- *SProperty*. It details all properties and attributes defined at the DIPM level with regard to the domain constraints. For instance, when dealing with RCES properties, we can consider *Powerconsumption* and *PhysicalPlatfromSize*. In addition, we propose to add *Quality* property to specify the target standard (IEEE certification,

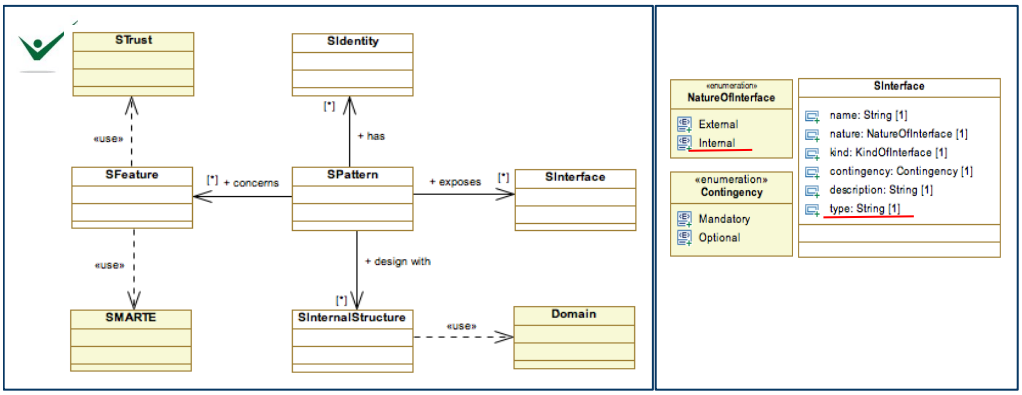

Fig. 4. S&D patterns at domain specific level

SIL, ...). The next section depicts more specific features values for the other domain independent features.

– *SInterfaces*. They are specialized by domain constraints in order to implement interaction with the application platform. For instance, at a low level, it is possible to define *External Interfaces* nature as links with software or hardware module for the cryptographic key management. In addition we propose to add *type* information to represent operations.

5 Illustration: Majority Voter Pattern

In the followings, an example will illustrate the approach point defined in the previous sections. For this issue, this example aims at providing a pattern from Industry Control Systems domain named *Majority Voter*[4]. The Majority Voter pattern is used in a critical application to perform a safety-related functionality regardless the presence of a transient fault. Such a pattern puts forward the following key points: firstly (Control Random Hardware Failures: Processing unit), the faults that affect the acquisition and analysis of the critical information and lead to incorrect results in the processing units must be detected and masked; secondly (Availability of information required by the Processing unit) The information required by the Processing unit must be available regardless the presence of a transient fault; thirdly (Reliability of the information required by the Processing unit) the information required by the Processing unit must be reliable regardless the presence of a transient fault.

For simplicity's sake, many functions of this use case have been omitted. We only detail the following use cases: (1) The Redundant_Architecture applies a hardware replication of the system to enhance the system availability/reliability in the presence of transient fault, and (2) the Decision_Maker must decide which system is properly

[4] Majority voter pattern is also known as Voter, a triple redundancy...

running. A normal scenario to use this pattern is the following: get the critical information from the application; analyze the information in each system of the Redundant_Architecture; send the data to the Decision_Maker; compare the information; send the valid data to the Actuator.

In the next subsections, we will describe the interfaces and the properties following the two abstraction levels (i.e., DIPM and DSPM) in order to support the two main use cases.

5.1 Representation at DIPM

Figure 5 illustrates the representation of Majority Voter pattern at DIPM. At this level, the pattern deals with a system which requires availability/reliability in the presence of transient faults. With regard to the interface, at this level the Majority Voter offers three External Interfaces:

- Redundant_Architecture: applies a hardware replication of the system to enhance the system availability/reliability,
- Decision_Maker: outputs the desired data regardless of whether one of the channels has a transient failure; the desired data is available regardless the presence of a transient fault;and
- Safety_Functionality: one can use this interface to configure the pattern.

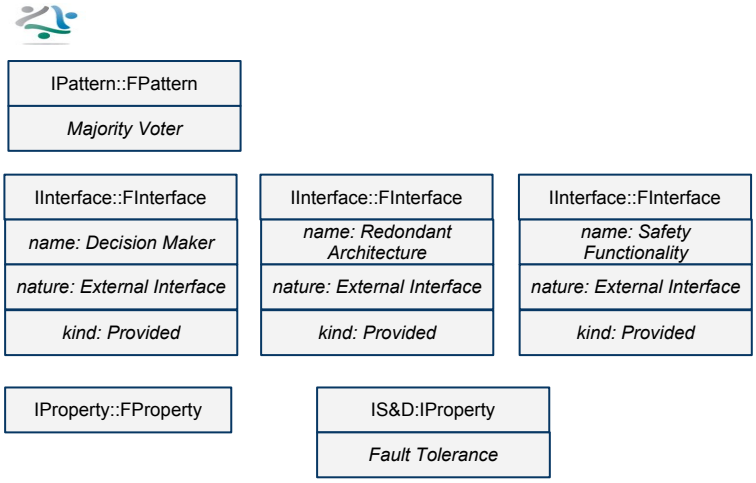

Fig. 5. Majority Voter Pattern at DIPM

5.2 Representation at DSPM

The interfaces must be adapted in order to match with the specific communication used in the domain. At this level, it is possible to refine these interfaces by defining their type as a set of operations. Moreover, it is possible to add new interfaces. For instance, an Internal Interface can be added like the *Determinism Of Information* and the *Integrity Algorithm* to interact with the target Safety_Critical_Embedded_Systems_Environment. Regarding to the properties, at the DIPM, we only specify a very generic properties. At this level, it is possible to refine these properties by defining the class of failures. Moreover, it is possible to add new properties. For instance, a RCES property can be added like the *Quality Standard*. Figure 6 illustrates the representation of Majority Voter pattern at DSPM.

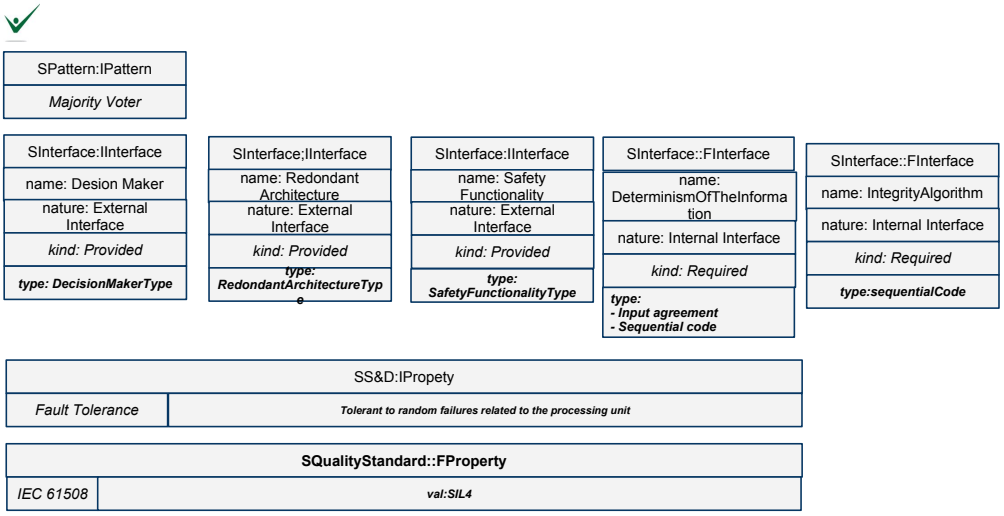

Fig. 6. Majority Voter Pattern at DSPM

6 Related Work

The design patterns are a solution model to generic design problems, applicable in specific contexts. Since their appearance, and mainly through the work of Gamma et al [10], they have attracted much interest. The supporting research includes domain patterns, pattern languages and their application in practice.

6.1 S&D Patterns

Several tentatives exist in the S&D design pattern literature [23, 11, 24, 6, 22]. 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.

Particularly, [23] presented a collection of patterns to be used when dealing with application security. The proposed catalog includes secure access layer, single access point, check point, etc.. The work of [11] reports an empirical experience, about the adopting 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. A survey of approaches to security patterns is proposed in [24].

Also, in developing fault-tolerant software applications, the use of patterns would lead to well structured applications. That is, [6] described an hybrid set of pattern to be 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,$\cdots$ In addition, the hybrid pattern structure can be constructed through recursive combination of N-Version programming and the others. The work addressed also the power of the technique through the support of the advanced software voting techniques. Extending this framework, [22] proposed a framework for the development of dependable software systems based on a pattern approach. They reused proven fault tolerance techniques in 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. The work is illustrated with an application to guide the self-repair of the system after the detection of a node crash.

6.2 Pattern Languages

To give a flavor of the improvement achievable bu using specific languages, we look at the pattern formalization problem. *UMLAUT* was proposed by Guennec et al. [1] as an approach that aims to formally model design patterns by proposing extensions to the UML meta model 1.3. They used OCL language to describe constraints (structural and behavioral). These constraints are defined on meta-models of specified UML elements in the form of meta collaboration diagrams. Mechanisms of association of these meta level diagrams to their instances level (instances of design patterns) are then defined. This allows to model design patterns accurately in UML language. This work is illustrated through two examples of design patterns: visitor and observer.

In the same way, Kim et al. [4] presented *RBML (Role-Based Meta modeling Language)*. The RBML is able to capture various design perspectives of patterns such as static structure, interactions, and state-based behavior. This language is based on the meta-modeling design patterns and offer $three$ specifications: Structural, Behavioral and Interactive. Each one is characterized by a kind of RBML meta-model: (1) SPS (Static Pattern Specifications) is a specification of structural design pattern which allows to express the static view, (2) IPS (Interaction Pattern Specification) represents

the design pattern in terms of possible interactions between different roles, (3) SIMP (StateMachine Pattern Specifications) can add to the specification of design pattern a point of view in behavioral to describe the various states in which it may lie in its execution.

Another issue raised in [8] and [5] is visualization. Eden et al. [8] presented a formal and visual language for specifying design patterns called *LePUS*. It defines a pattern in an accurate and complete form of formula with a graphical representation. A diagram in LePUS is a graph whose nodes correspond to variables and whose arcs are labeled with binary relations. With regard to the integration of patterns in software systems, the *DPML (Design Pattern Modeling Language)* [5] allows the incorporation of patterns in UML class models.

6.3 S&D Modeling Languages

Many studies have already been done on modeling security in UML. [14] presents an extension UMLsec of UML that enables to express security relevant information within the diagrams in a system specification. UMLsec is defined in form of a UML profile using the UML standard extension mechanisms. [16] presents a modeling language for the model-driven developmentofsecure, distributed systems based on UML.Their approach is based on role-based access control with additional support for specifying authorization constraints. SecureUML is a modeling language that defines a vocabulary for annotating UML-based models with information relevant to access control.

In [12], we proposed a methodology associating model-driven approach and component based development to design distributed applications that has fault-tolerance requirements. UML based modeling is used to capture application structure and related non-functional requirements thanks to the complementary profile named FT profile which is composed of an extension of a subset of QoS&FT and uses NFP (Non Functional Properties) sub-profile of MARTE[17] (profile for Modeling and Analysis of Real-Time Embedded systems). Stereotypes dedicated to fault-tolerance specify the fault-detection policy, replication management style, replica group management. From this model we generate descriptor files (according to Deployment and Configuration standard (D&C)) to build bootcode (static deployment) which instantiates, configures and connects components and to load configured components. Within this process, component replication and FT properties are declaratively specified at model level and are transparent for the component implementation.

In addition to the above, the recently completed *FP6 SERENITY* project has introduced a new notion of S&D patterns. SERENITY's S&D patterns are precise specifications of validated security mechanisms, including a precise behavioral description, references to the S&D provided properties, constraints on the context required for deployment, information describing how to adapt and monitor the mechanism, and trust mechanisms. Such validated S&D patterns, along with the formal characterization of their behavior and semantics, can also be the basic building blocks for S&D engineering for embedded systems. [21] explains how this can be achieved by using a library of precisely described and formally verified security and dependability (S&D) solutions, i.e., S&D classes, S&D patterns, and S&D integration schemes.

6.4 Positionning

While many S&D patterns have been designed, still few works propose general techniques for S&D patterns. As soon as most of them uses the same generic concepts to characterize patterns, there is not a real consensus about what is a pattern. Furthermore, the term pattern is often ambiguous because it is used to encode the solution of a recurrent problem and to deal with a model instead of pattern implementation. In software engineering, design patterns are considered as effective tools for the reuse of specific knowledge. However, there is still a gap between the development of the system and the pattern information.

For the first kind of approaches [10], design patterns are usually represented by diagrams with notations such as UML object, most often accompanied by textual descriptions and examples of code to complete the description. Furthermore their structure is rigid (Context, Structure, Solution, etc.). Unfortunately, the use and / or application of a pattern can be difficult or inaccurate, in effect; the existing descriptions are not formal definitions and sometimes leave some ambiguity about the exact meaning of patterns. There are some promising and well-proven approaches [7] based on Gamma et al. However this kind of techniques do not allow to reach the high degree of pattern structure flexibility which is required to reach our target. is a framework that aims the specification of design patterns. The visualization technique promoted by LePUS [8] is interesting but the degree of expressivity proposed to model design a pattern is too restrictive.

UMLsec [14] (approach based on modeling security in UML) and our proposal are not in competition but they complement each other by providing different view points to the secure information system. In concept, our modeling framwork is similar to the one proposed in SERENITY project. However, Nevertheless the pattern structure is rigid (a pattern is defined as quadruplet) and consequently is not usable to capture specific characteristics of S&D patterns. We note, however, that SERENITY proposes several levels of abstraction to bridge the gap between abstract solution and implementation but not to get a common representation of patterns for several domains.

As a side remark, note that our goal is encode pattern following different levels of abstraction to get a common representation of patterns for several domains. That is, we propose an even high level abstraction to represent S&D patterns to capture several facets of security and dependability in the different domain of RCES, not an implementation of a specific solution.

7 Discussion and Conclusion

In this paper we proposed a model-based patterns to deal with security and dependability for resource constrained embedded systems with trust requirements. This work will be used as brick to build a trusted applications through a model driven engineering approach.

Our proposition is based on several levels of abstraction: meta, generic and domain specific levels. The aim at the meta level is to capture, at high level, a set of generic properties by determining in advance if the artifact (e.g. pattern and interfaces) has or

uses a certain kind of generic properties. We use software component structure vision (such as defined in CBSE) to capture several facets of security and dependability in the different domain of RCES. The result flexible structure allows to use the pattern in RCES on easy way. In order to illustrate our findings in the context of embedded systems, a demonstrative case study that has dependability requirements (majority voter) is examined: industrial control system domain. For instance to use a redundancy with voter protocol to tolerate transient failures, at generic level we define instances of artifacts to encode majority voter pattern including interfaces, properties,.... Then, the domain specific level allows to specialize it to the industrial control system domain with more dedicated information.

The key is then to show that the major sectors of RCES dealing with security and dependability become covered by our approach. This result raises new and previously unanswered questions about general techniques to encode S&D patterns. We believe that this result is of particular interest to build a trusted computing engineering discipline that is suited to a number of sectors in resource constrained embedded systems.

The next steps are primarily to implement other patterns including those for security and dependability to build a repository of S&D patterns. This will used as a suited tool to examine the compliance and the completeness of our meta-model. Then, we plan to study frameworks to verify and validate patterns following the different level of abstraction in order to build a proved pattern in an unified way. For instance, we aim at ensuring that the pattern is still correct with regard to its intention after the execution of transformation rules in order to specialize it.

Another objective for the near future is to provide guidelines concerning both the implementation of S&D patterns, to catalog them and to integrate them in a MDE process.

References

1. G. Sunyé A. L. Guennec and J-M. Jézéquel. Precise modeling of design patterns. Springer-Verlag, 2000.
2. C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*, volume 2 of *Center for Environmental Structure Series*. Oxford University Press, New York, NY, 1977.
3. G. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: a system of patterns*, volume 1. John Wiley and Sons, 1996.
4. S. Ghosh D-K Kim, R. France and E. Song. A uml-based meta-modeling language to specify design patterns. 2004.
5. J. Grundy D. Mapelsden, J. Hosking. Design pattern modelling and instantiation using dpml. In *CRPIT '02: Proceedings of the Fortieth International Conference on Tools Pacific*, pages 3–11. Australian Computer Society, Inc., 2002.
6. F. Daniels. The reliable hybrid pattern: A generalized software fault tolerant design pattern. 1997.
7. B. P. Douglass. *Real-time UML: Developing Efficient Objects for Embedded Systems*. Addison-Wesley, 1998.
8. A. H. Eden E. Gasparis, J. Nicholson. Lepus3: An object-oriented design description language. In *In: Gem Stapleton et al. (eds.) DIAGRAMS, LNAI 5223*, page 364–367, 2008.

9. F. Fondement and R. Silaghi. Defining model driven engineering processes. In *in Proceedings of WISME*, 2004.
10. E. Gamma, R. Helm, R. E. Johnson, and J.Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
11. V. Di Giacomo and al. Using security and dependability patterns for reaction processes. pages 315–319. IEEE Computer Society, 2008.
12. B. Hamid, A. Radermacher, A. Lanusse, C. Jouvray, S. Gérard, and F. Terrier. Designing fault-tolerant component based applications with a model driven approach. In *The IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems(SEUS)*, Lecture Notes in Computer Science, pages 9–20. Springer, 2008.
13. T.A. Henzinger. Two challenges in embedded systems design: Predictability and robustness. *Philosophical Transactions of the Royal Society A*, 366:3727–3736, 2008.
14. J. Jürjens. Umlsec: Extending uml for secure systems development. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 412–425, London, UK, 2002. Springer-Verlag.
15. H. Kopetz. The complexity challenge in embedded system design. In *ISORC*, pages 3–12, 2008.
16. T. Lodderstedt, D. Basin, and J. Doser. Secureuml: A uml-based modeling language for model-driven security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 426–441, London, UK, 2002. Springer-Verlag.
17. OMG. Omg. a uml profile for marte: Modeling and analysis of real-time embedded systems,beta 2. http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf, June 2008.
18. S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady. Security in embedded systems: Design challenges. *ACM Trans. Embed. Comput. Syst.*, 3(3):461–491, 2004.
19. D. Schmidt. Model-driven engineering. *in IEEE computer*, 39(2):41–47, 2006.
20. M. Schumacher. *Security Engineering with Patterns - Origins, Theoretical Models, and New Applications*, volume 2754 of *Lecture Notes in Computer Science*. Springer, 2003.
21. D. Serrano, A. Mana, and A-D Sotirious. Towards precise and certified security patterns. In *Proceedings of 2nd International Workshop on Secure systems methodologies using patterns (Spattern 2008)*, pages 287–291. IEEE Computer Society, September 2008.
22. M. Tichy and al. Design of self-managing dependable systems with uml and fault tolerance patterns. pages 05–109. ACM, 2004.
23. J. Yoder and J. Barcalow. Architectural patterns for enabling application security. In *Conference on Pattern Languages of Programs (PLoP 1997)*, 1998.
24. N. Yoshioka, H. Washizaki, and K. Maruyama. A survey of security patterns. *Progress in Informatics*, (5):35–47, 2008.
25. R. Zurawski. Embedded systems. CRC Press Inc, 2005.