Engineering secure systems: Models, patterns and empirical validation

Brahim Hamid, Donatus Weber

Engineering secure systems: Models, patterns and empirical validation

Brahim Hamid[a,*], *Donatus Weber*[b]

[a] *IRIT, University of Toulouse, 118 Route de Narbonne, Toulouse Cedex 9 31062 France*
[b] *Chair for Embedded Systems, University of Siegen, Hoelderlinstrasse 3, D-57076 Siegen, Germany*

A B S T R A C T

Several development approaches have been proposed to handle the growing complexity of software system design. The most popular methods use models as the main artifacts to construct and maintain. The desired role of such models is to facilitate, systematize and standardize the construction of software-based systems. In our work, we propose a model-driven engineering (MDE) methodological approach associated with a pattern-based approach to support the development of secure software systems. We address the idea of using patterns to describe solutions for security as recurring security problems in specific design contexts and present a well-proven generic scheme for their solutions. The proposed approach is based on metamodeling and model transformation techniques to define patterns at different levels of abstraction and generate different representations according to the target domain concerns, respectively. Moreover, we describe an operational architecture for development tools to support the approach. Finally, an empirical evaluation of the proposed approach is presented through a practical application to a use case in the metrology domain with strong security requirements, which is followed by a description of a survey performed among domain experts to better understand their perceptions regarding our approach.

Keywords:
Security
System engineering
Pattern
Meta-modeling
Model driven engineering

1. Introduction

System and software security engineering (Anderson, 2008; Barnabe et al., 2011; Devanbu et al., 2000) has become a crucial business aspect because organizations are completely dependent on computer-based systems and invest substantial resources in maintaining them. Standards are available for securing IT systems, such as NIST 800-60, and Control Systems (ICS), such as NIST 800–82. Although they give little guidance to software engineers on how to implement them, they should be applied from the early stages of the conception of a system. Most work must be done manually because only a few tools are available to aid in the implementation. This causes extensive work and incurs substantial extra costs. Thus, there is a need to support the engineering of secure system processes with as much automation as possible. Therefore, developers of these systems need to "design for security". This includes defining the current structure of the system, i.e., the system architecture, finding abstract risks and concrete vulnerabilities, and implementing the appropriate countermeasures to mitigate risks and vulnerabilities to meet the security requirements of these systems. Our contribution to this challenge is to make the system security engineering process manageable and understandable through novel methods and tools that ensure that system security solutions are built by design.

Security experts, practitioners and researchers from different international organizations, associations and academia have agreed that for security, "it's not just the code" (Fernandez, 2013; M. Howard, 2007; Neumann, 2004). The most popular and well-known software security vulnerabilities are design issues, particularly architecture design issues. From the system developer perspective, security issues need

* Corresponding author.
E-mail address: hamid@irit.fr (B. Hamid).
https://doi.org/10.1016/j.cose.2018.03.016

to be identified early in the first development steps and at the highest levels, primarily in the architecture design stage, where their semantics are clear. When security requirements are determined, architecture and design activities are conducted using modeling techniques and tools for higher quality and seamless development. The integration of security features using this approach requires high expertise for developing the architecture and design and the availability of both application-domain-specific knowledge and security expertise. Because few experts with this diverse set of experiences exist, capturing and providing this expertise by means of security patterns (Anwar et al., 2006; Hafiz et al., 2007; Schumacher, 2003; Yoder and Barcalow, 1998) can enhance system development by integrating them into different stages of systems engineering. Therefore, security solutions are described as security patterns, and the application-domain specific use of these patterns is provided in terms of security technological products that are already established in that domain.

In this paper, we present a model-based approach for engineering secure systems that uses *patterns* to represent security solutions and knowledge, which fosters reuse. This work is conducted within the context of a model-based security and dependability research project, and our collaboration with security-critical system suppliers suggested a need for this work. The security solutions used by security-critical system developers are based on the application domain and occasionally on the software development environment, including the design and coding stages. There is a need to link these concepts to security. The lack of appropriate links between application domain concepts and security concepts poses three main challenges. First, the security engineer must re-engineer existing solutions. Second, the application designer may not understand domain-specific security solutions. Finally, it is difficult for a system developer to guarantee the availability of security solutions to cover the security requirements of the targeted application using application domain concepts.

To provide a concrete example, we use smart meter systems. In these systems, there are a number of subsystems working together to communicate and exchange the up to date sensing and measurement of energy consumed with the smart grid. Within these large systems, we focus only on a smart meter gateway as the communication unit of modern smart meters. A gateway is capable of connecting to several meters for different commodities, such as electricity, gas, water or heat, and communicating with households and other remote entities, such as regulations. Smart meters and their gateways generate security and safety relevant, as well as privacy sensitive data and transmit them over possibly insecure networks. Therefore, a special kind of protection is required for this data.

To address the above problem, we promote a new methodology for system engineering using a pattern as its first-class citizen: Pattern-based System and Software Engineering (PBSE). We use MDE (Atkinson and Kühne, 2003) abstraction mechanisms to define and handle security patterns through a metamodel that unifies those concepts. Moreover, we use MDE transformation mechanisms (France and Rumpe, 2007; Selic, 2003) that can adapt and generate different representations, where patterns are clearly related to domain models. Such an MDE-based approach utilizes domain-specific modeling languages (DSMLs) (Gray et al., 2007) built on an integrated repository of modeling artifacts that function as a group, where a pattern is at the heart of development - its role should be specified in all lifecycle stages of development. In addition, we aim to provide formal security semantics for pattern-based system models. As a result, patterns can be used as bricks to build applications through a model-driven engineering approach. In this work, Eclipse Modeling Framework Technology (EMFT) [1] is used to build the support tools for our approach. All metamodels are specified using the EMF. The repository is implemented using the approach described in (Hamid, 2017), which is based on the Eclipse CDO [2] framework. In our work, the target domain development environment is IBM Rational Rhapsody [3], and the descriptions of the model transformations are based on the QVT operational language (OMG, 2011). However, our vision is not limited to the EMF platform. Other modeling tools conforming to the requirements of Section 5 can also be used.

The main goal of this work is to define a modeling and development framework to support the specifications, definitions and adaptation of a set of modeling artifacts to assist the developers of secure applications. The patterns that are at the heart of our system engineering process reflect design solutions at the domain-independent and domain-specific levels. The envisioned modeling framework offers an integrated conceptual design for the specification and development of security patterns and a concrete and coherent methodology for the development of software systems based on security patterns. We provide evidence of its benefits and applicability through the example of a representative industrial case, i.e., the smart meter gateway application.

We have previously studied various facets of PBSE to engineer secure and dependable systems. Our prior work includes modeling languages for the specification of security and dependability patterns (Hamid et al., 2011), design and implementation of a reuse model repository (Hamid, 2014). Further, the basic formulation of the need to have a strong approach for supporting the engineering of secure systems has been previously published in a research paper in the international journal of Innovations in Systems and Software Engineering (Hamid et al., 2016). This article brings together and extends the ideas described in the earlier papers and presents a holistic approach to the modeling of pattern-based software systems with strong security requirements. Specifically, we provide a more comprehensive and complete description of our approach (Section 4) and tool support (Section 5), along with substantial new validation to show the feasibility and usefulness of our approach (Section 6).

The remainder of the paper is organized as follows. An overview of the challenges faced in engineering secure systems in the context of industry standards and practices is presented in Section 2. Section 3 provides a motivating example that models a software architecture for a web application. In Section 4, we present our approach for supporting the pattern-based system and software security engineering. Section 5 describes the architecture of the tool suite and presents an

[1] https://eclipse.org/modeling/emft/.
[2] http://www.eclipse.org/cdo/.
[3] http://www-03.ibm.com/software/products/en/ratirhapfami.

example implementation. In Section 6, we present an empirical evaluation of our approach through the TERESA metrology case study and a survey. Section 7 presents experimental results in the context relevant to security goals and threat analysis. Then, Section 8 discusses our contribution. Section 9 compares our work with related work. Finally, Section 10 concludes and sketches future work directions.

2. Generalities and background

The engineering of secure systems with security requirements typically requires the certification of such products according to generic standards (e.g., Common Criteria (CC) 15408-1 (ISO/IEC, 2007), ISO 31000 (Dali and Lajtha, 2012; ISO, 2009)) or domain-specific standards (e.g., NIST 800-60 (Stine et al., 2008), ISO/IEC 27001 (ISO, 2013), Protection Profile (PP) (BSI, 2014)). They support system engineers in the development of secure systems recommending a set of techniques and measures to achieve the desired level of security of an application. For instance, the CC standard provides support when building a secure system by offering classes of functional security components to address well-known security principles (e.g., authenticity, encryption, and audit) and a set of guidelines to support security experts. Through the aforementioned PP standard it is possible to add domain specific to the general approach in extending and refining requirements and knowledge to the targeted domain (e.g., encryption standards, hash functions, SSL, and TPM).

The engineering of information systems with security has been well established via standards and general methods. However, they provide little guidance to software engineers on how to implement them. Most work must be done manually, as only a few tools are available to aid in the implementation. Most of these techniques and measures are provided in the form of design decisions targeting one stage of the development lifecycle without effective realization. They should be applied from the very beginning stage of the conception of the system. In addition, the format in which the techniques and measures are presented must be improved to ensure that the provided solutions are storable, reusable and appropriate for the automation of software development and analysis. Furthermore, the cost-effective development and certification of the targeted products is a challenge, and the reusability of proven solutions enables cost and time-to-market reductions. Our work aims to provide new methodological tool support that enables these solutions (techniques and measures) to be reused for development within the same application domain or in a cross-domain scenario. As described below, a subset of techniques and measures proposed in the standards can be used to define several security patterns.

2.1. *Model-based software systems security engineering*

A general methodology for developing security-critical software using models has been proposed in (Jürjens, 2001; Jürjens, 2006). It uses a UML extension (profile) called UMLSec to include security-relevant information in the existing model and diagrams. The approach is supported by extensive automated tool support for performing a security analysis of the UMLSec models against security requirements and has been used in a variety of industrial projects (J. Jürjens and R. Rumm, 2008). We can also cite two other system design model-driven software security engineering processes based on UML profiles: SecureUML (Lodderstedt et al., 2002) and SecureMDD (Moebius et al., 2009). SecureUML provides a UML profile based on role-based access controls (RBAC), enabling specification of access control in the overall system design. This information can be used to generate access control infrastructures, helping the developers to improve productivity and the quality of the system under construction. SecureMDD proposes a methodology that enables the generation of platform-specific models (e.g., JavaCard) from a high-level stereotyped UML model. In addition to guidelines for modeling security aspects, the framework offers verification based on a formal approach for the produced models (Grandy et al., 2006). In addition to the above approaches, Lee et al. (Lee et al., 2002; 2000) propose an integration model for integrating security engineering approaches into software lifecycle standards, mapping the concepts of the software lifecycle (IEEE 12207) to security engineering concepts (a set of concepts collected from various security engineering approaches (Lee et al., 2000)). The approach attempts to provide an understanding to stakeholders of where and when security activities intervene and interact with standard process lifecycle activities. In the development of secure systems, Basin et al. (Basin et al., 2009) use a metamodel called SecureUML+ComponentUML, which combines SecureUML (Lodderstedt et al., 2002) and ComponentUML (a system design modeling language for component-based systems). This metamodel is used to model security design models and security scenarios starting from an informal security policy. A pattern-oriented approach for modeling secure component-based software systems combining patterns and UMLsec (Jürjens, 2006) was presented in (Schmidt and Jürjens, 2011).

2.2. *Pattern-based software systems security engineering*

Patterns are encapsulated solutions to recurrent system problems and define a vocabulary that concisely expresses requirements and solutions as well as provide a communication vocabulary for designers (Gamma et al., 1995; Henninger et al., 2007). Patterns are triples that describe *solutions* for commonly occurring *problems* in specific *contexts*. Pattern-based development has recently gained more attention in software engineering by addressing new challenges that had not been targeted in the past (Henninger et al., 2007). The idea of design patterns was introduced by an architect (Urbanist), Christopher Alexander (Alexander et al., 1977). The first objective was to enhance architectural quality, beauty, elegance and harmony to avoid dehumanization of the living environment. In *GoF* (Gamma et al., 1995), a design pattern extracts the key artifacts of a common design structure that renders it useful for creating a reusable object-oriented design. We now recall the definition of the term "security pattern" that we found in the literature (Schumacher, 2003).

Definition 1 (security pattern)**.** A security pattern describes a particular recurring security problem that arises in specific

contexts and presents a well-proven generic scheme for its solution.

Adapting the definitions of Schumacher (2003), we propose the following:

Definition 2 (System of security patterns)**.** A system of security patterns is a collection of security 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 objective.

Security patterns enable the development of secure applications and liberate the developer from having to address security-related technical details. MDE (Atkinson and Kühne, 2003; Selic, 2003) also provides a very useful contribution to the design of security-critical systems (Basin et al., 2009; 2006; Jürjens, 2006; Lucio et al., 2014) because it reduces the time/cost required for understanding and analyzing system artifact descriptions due to the abstraction mechanisms. It also reduces the development process cost due to the generation mechanisms. Hence, security pattern integration must be considered during the MDE process.

2.3. *Security analysis and evaluation*

A typical evaluation approach in software system security engineering is to compare an implementation with a comprehensive list of categories agreed upon by an expert community. An example is the OWASP (Open Web Application Security Project) list (OWASP, 2017b). On the one hand, the analysis discovers potential risks and areas for improvement; on the other hand, it can raise confidence in the chosen architectural approaches.

The current practice to formulate security statements is given through expressing attacker capabilities to e.g. gain access to a protected data from a message observation (i.e., negative statements). Microsoft for example uses a threat taxonomy from the attacker's perspective called STRIDE [4]. To define the security architecture of the system, we need an analysis of the possible threats; security patterns can then be introduced to stop or mitigate them (Fernandez, 2013; S.T. Halkidis et al., 2008). As opposed to this, the property-based approaches(Fuchs et al., 2010; J. Jürjens and R. Rumm, 2008) express security statements in terms of a set of desirable security properties (i.e., positive statements) using common taxonomies such as CIA [5]. Patterns then are introduced according to expected security properties.

During the security application design time, formal modeling of patterns can prove some of the properties of a pattern solution (security mechanisms), which may require a specialized and costly verification process (e.g., based on formal methods (Hamid et al., 2016; Paulson, 1996)). At system design time, we can evaluate the security risk of systems that used specific security patterns by seeing how well they handled a set of threats by a simple matching of threats to patterns (Fernandez, 2013). Often, an attack on a system results in the not-fulfillment of specific security properties. Therefore, threats can be stopped by the fulfillment of security properties. If we have a pattern for each security property, we can consider the system secure at the model level. Since then, it is still remains vulnerabilities at code level. However, attacks at code level are reduced by an appropriate architectural design.

Further, for systems requiring standardization and certification, we can rely on existing security risk analysis methods such as EBIOS (McDonald et al., 2013), CORAS (Braber et al., 2007) and security-HAZOP (Srivatanakul et al., 2004). In these methods, a threat and risk analysis is executed using methods such as the threat modeling with attack trees, as described in (Schneier, 1999). The output of these methods is a set of recommendations and guidelines to detect possible risks, evaluate them and then mitigate them. Patterns enable the implicit application of policies and in the same time pattern is a systematic approach to describe best practices. Further, pattern can describe standards and regulations in a precise way and make them more understandable and usable.

3. A motivating example

The example is based on an OWASP example of a college library website (OWASP, 2017a). Fig. 1 shows the overall architecture description of the web application. It consists of a client, a web server and a database server software components. The website provides online services for searching for and for requesting books. The users are students, college staff and librarians. Staff and students will be able to log in and search for books, and staff members can request books. Librarians will be able to log in, add books, add users, and search for books. We use UML class diagram to describe the high level architecture model of the web application, where software components are represented by classes, and relationships between these components are represented by associations.

Fig. 1 depicts the corresponding software architecture. There are implicit security design decisions, because internal and external connections use public or private networks. There is also an ad-hoc subcomponent for "authorization" that groups three entities (*Webserver, Database and Administrator*) that collaborate in order to ensure that the application has clearly defined the user types and the rights of said users. Nevertheless, several controls related to security are outlined in Table 1. These controls may be found in each respective standard of each domain (BSI, 2014; ISO, 2013; Stine et al., 2008; OWASP, 2017a).

Fig. 2 shows the class diagram that adds security controls of Table 1. Each one represents a specification (blue boxes). However, effective realizations of these controls are not modeled in the UML class diagram; they may be subject to certain changes and/or adaptations (new security solutions, deletions, modifications of realization, for instance), verifications (formal and empirical, for instance) and reuse (in the same domain or across domains, for instance) while the structure of the main software architecture can be maintained.

Fig. 3 shows the class diagram that adds new components to realize the security controls of Table 1. Each one represents a pattern (red boxes), which secure software developers,

[4] STRIDE stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.

[5] CIA stands for Confidentiality, Integrity and Authenticity.

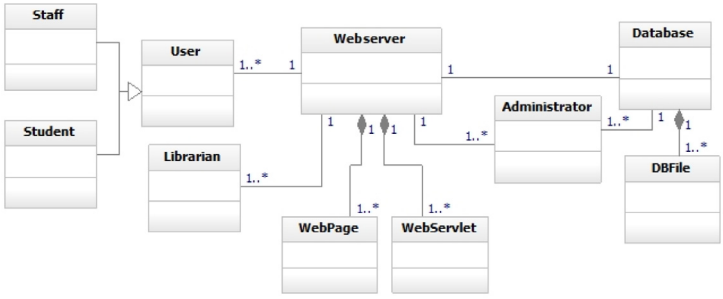

Fig. 1 – A motivating example.

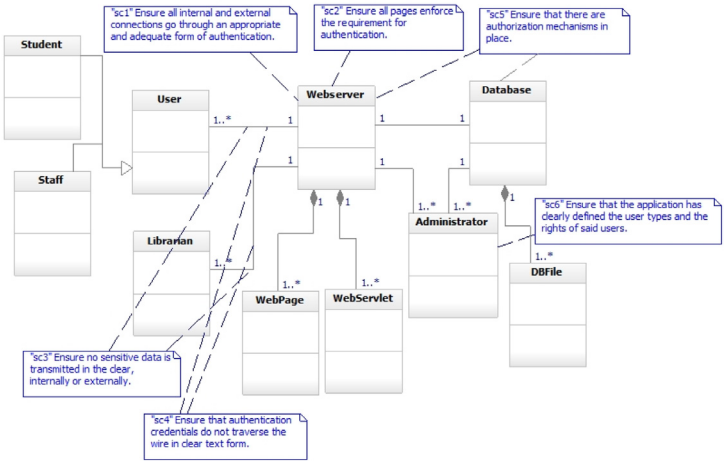

Fig. 2 – Motivating example with security controls.

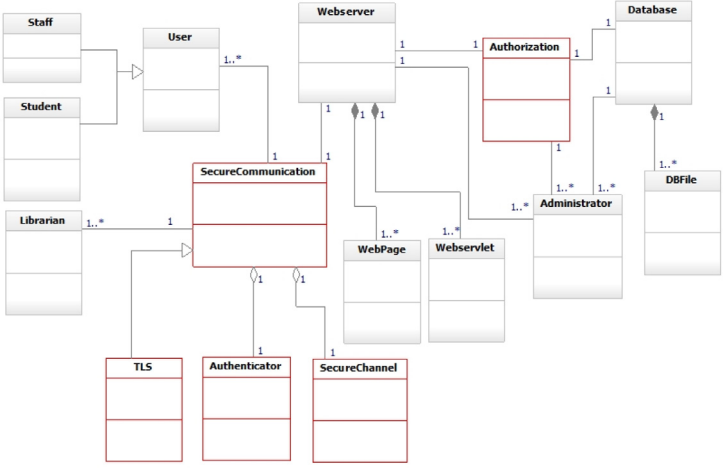

Fig. 3 – Motivating example with security patterns.

mainly architects would like software modeling and analysis languages may easily express.

At this point, several questions were raised: If existing modeling languages (Basin et al., 2009; J. Jürjens and R. Rumm, 2008; Moebius et al., 2009) provide support for engineering secure systems, how can we define these patterns in terms of their constructs? Must we modify and overload the existing software engineering process to define these patterns? Are there alternatives to specify these patterns while maintaining the overall structure of the software engineering process? Do the existing software modeling languages have enough expressiveness to seriously address these issues? What ideas are researchers proposing to incorporate security in model-based software system engineering ? What advantages and disadvantages have their proposals? Further, we attempt to add more formality to improve parts of the system design. In

Table 1 – List of security controls (OWASP, 2017a).

Security property	#	Security controls
Authenticity	sc1	Ensure all internal and external connections go through an appropriate and adequate form of authentication.
	sc2	Ensure all pages enforce the requirement for authentication.
Confidentiality	sc3	Ensure no sensitive data is transmitted in the clear, internally or externally.
	sc4	Ensure that authentication credentials do not traverse the wire in clear text form.
Authorization	sc5	Ensure that there are authorization mechanisms in place.
	sc6	Ensure that the application has clearly defined the user types and the rights of said users.

our context, we have identified two dimensions: a global system viewpoint (e.g., does the system provide a sufficient security level?) and an individual functional viewpoint (e.g., is a specific security function effective?).

Therefore, there are two main prerequisites to defining the pattern-based software system security engineering methodology. The first is that it must be compatible with current development processes. The objective is not to change the habits of engineers; instead, easing the acceptance of the approach in industry is the goal. The second prerequisite is that it must be flexible enough to adapt to other specific processes in other domains. We seek a solution based on the reuse of software subsystems, i.e., so-called security patterns that have been pre-engineered to adapt to a specific domain. The approach described in Section 4 uses MDE techniques to handle the issues described above.

4. Approach

The proposed approach (Fig. 4) is composed of six main steps (the numbers in parentheses correspond to the numbers in Fig. 4). Step 1 is responsible for creating the conceptual model of security patterns. The resulting conceptual model is used to build a DSML to specify security patterns (step 2). The security expert, with the system and a software engineering expert, uses this DSML to define security patterns (step 3). Then, a domain process expert creates and adapts the security patterns into a version that is suitable in its system development process (step 4). An example is adaptation for compliance with an appropriate standard. Moreover, in the context of our work, certain patterns have a meaningful representation only for a certain domain. We then develop and apply appropriate transformations of a pattern representation in a suitable format for the development environment (step 5). Pattern instantiation as the initial activity to apply a pattern is performed during step 4 and step 5. Finally, the domain engineer reuses the resulting adapted and transformed patterns for the given engineering environment (development platform) to develop a domain application (step 6). Pattern integration in the design activity is applied during this step. In addition, we attempt to use formal description (e.g, formal modeling) of security and architecture for the purpose of the development of an accurate analysis, for evaluation and/or certification.

The first two steps (1 and 2) are performed once for a set of domains. The inputs of these steps are expertise, standards and best practices from the security expert. Step 3 is performed once for a set of domains. Step 4 is performed once per application domain. Performing Step 3 requires knowledge of security engineering, whereas Step 4 requires knowledge of both security engineering and the system development process for a specific application domain. Step 5 is performed once for each development environment. Step 6 is performed once for every system in the application domain. This step requires the availability of knowledge on the specific targeted system and dedicated tools that are customized for a given development platform. In the rest of this section, we present detailed descriptions of the six steps in our approach.

4.1. *Step 1: Conceptual model of security patterns*

At the beginning of this paper, we indicated the need for an explicit interpretation of security in architecture design and motivated why patterns are a useful tool for designing secure systems architectures. We achieve this task in the first step of our approach via the creation of a conceptual model of security patterns. A conceptual model of security patterns provides a common understanding of all concepts employed in this paper to ensure a precise description of the addressed problems and solution approaches.

A conceptual model of a security pattern should capture the main concepts and relationships to describe the security pattern models in the context of different standards and domain-specific practices. To maintain the current audience and awareness, we retain part of the terminology defining sections in the template to document patterns (Alvi and Zulkernine, 2011; Buschmann et al., 2007). We employ UML class diagrams to describe the conceptual model, where concepts are represented by classes, concept attributes are represented by class attributes, and relationships between concepts are represented by associations. When an attribute's value belongs to a predefined set of possible values, we use enumerations. Package notation is used to create groupings of concepts.

A graphical representation of the concepts and relationships from the excerpt is given in Fig. 5. To improve understanding and readability, the attributes of the different concepts and the links between the concepts are not described. We note that the model in Fig. 5 is a partial representation of the concepts and relationships relevant to the pattern definition and usage.

The `context package` contains all concepts for describing the environment in which the pattern will be built and employed, including security policy statements, the system development lifecycle, and the type of system assets to protect (e.g., applications, distributions, and data). The `problem package` contains all concepts that are employed for describing the security problem, including the threat catalog, threat scenario, vulnerabilities, types of attacks, security objectives, functional security requirements and other forces. The `solution package` regroups security technology concepts, such as best practices, controls, safeguards, countermeasures,

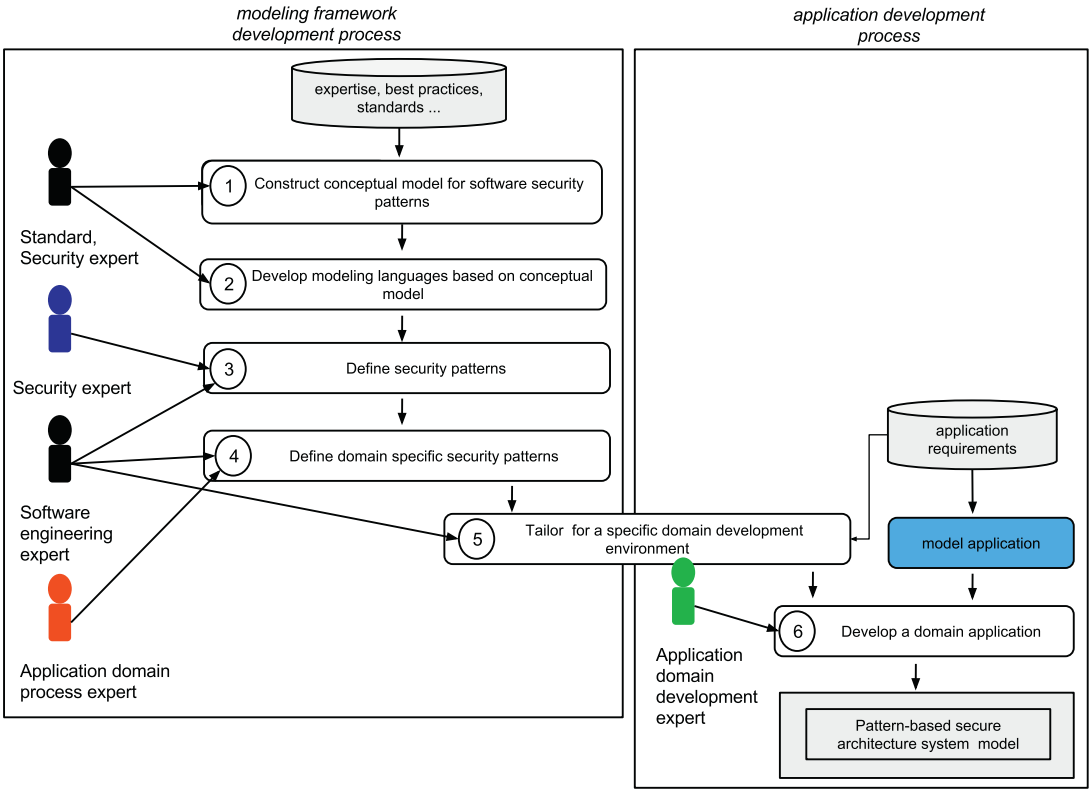

Fig. 4 – Methodology for the creation of the PBSE modeling framework.

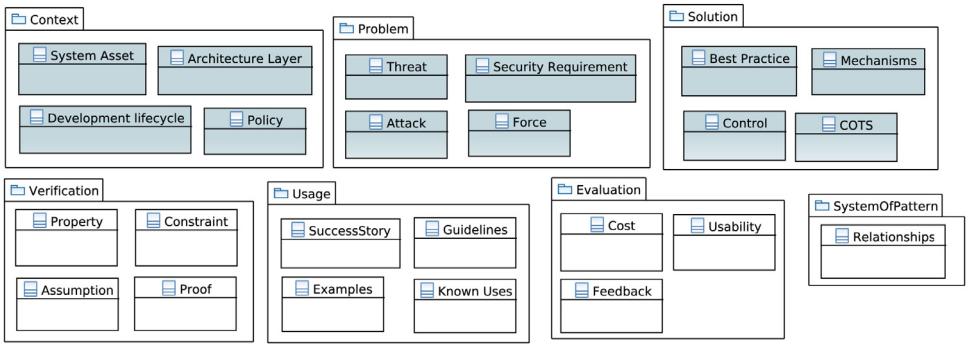

Fig. 5 – Security pattern concepts and their relationships.

design solutions, security mechanisms and COTS (Commercial of the Shelf) components. The `usage package` regroups all activities for integration-application in the designs and classification information to support search mechanisms that surpass keyword-based search and retrieval, including guidelines for pattern application, known uses and examples. The `system of patterns package` contains all concepts for describing the collection of patterns and patterns' relationships with other patterns, including composition, dependencies and conflicts. The `verification package` regroups concepts to check security properties, which ensures the quality of the solution including properties, assumptions and constraints. The `evaluation package` regroups concepts for evaluating patterns, including concepts to describe the impact on other architecture quality attributes, including performance, cost, usability and feedback.

These concepts and their related observations are employed as a basis for our conceptual pattern modeling language (refer to Section 4.2).

4.2. *Step 2: Creation of a DSML from a conceptual model of security patterns*

In this section, we emphasize software patterns as a way to design a secure software architecture (Fernandez, 2013), which builds on a semi-component patterns representation. In software engineering, the separation of concerns promotes the separation of general-purpose services from

implementations. In our context, we target the separation of the general purpose of a pattern from the mechanisms employed to implement the pattern. This is an important issue for understanding the use of patterns in the scope of security engineering. The layer in which patterns and their related mechanisms are integrated is dependent on the assurance of a client in the services of other concerned layers. Security patterns are defined from a platform-independent perspective (i.e., they are independent of their dedicated implementation mechanisms); they are consistently expressed with domain-specific models. Consequently, they are much easier to understand and validate by application designers in a specific area. To capture this vision, we introduce the concept of the *domain perspective*, where a security pattern at the domain-independent level exhibits an abstract solution without specific knowledge of how the solution is implemented with regard to the application domain. The objective is to reuse the domain-independent model security patterns for several application domains and enable them to customize these domain-independent patterns with their domain knowledge and/or requirements to produce their domain-specific artifacts.

To model security patterns, a modeling language was developed based on the conceptual model created in Step 1. The System and software Engineering Pattern Metamodel (SEPM) is constructed based on an additional analysis of different standards from different domains that focus on software-based systems. The set of concepts that were employed to develop the modeling language to capture these two levels of details to represent security patterns were identified: abstract pattern concepts to define a pattern at the domain-independent level (DIPM) and concrete pattern concepts to define a pattern at the domain-specific level (DSPM). For instance, abstract pattern concepts include *policies* and *best practices*, whereas concrete pattern concepts include *domain best practices* and *mechanisms*.

For our purpose, we propose using a well-known approach in MDE: a DSML. This approach is useful because we are storing a library of design patterns in a common repository and providing one or more adaptations of each pattern to target several application domains, e.g., the metrology industry, and different development environment domains, e.g., UML. However, our vision is not limited to DSML. For example, in (Radermacher et al., 2013), we defined a UML profile under the UML papyrus tool [6] to specify patterns.

4.2.1. *Informal description of the motivating pattern example*

The problem addressed in our example is how to ensure that the data transmitted over any public network is secure in transit, particularly how to guarantee data authenticity. We show the feasibility of our approach through the example of Secure Communication Pattern (SCP). On the domain-independent level, this pattern uses abstract send and receive actions and abstract communication channels that are assumed to provide authenticity. However, on the domain-specific level, SCPs are slightly different in the application domain. A system domain may have its own mechanisms, means and protocols that can be employed to implement this pattern include SSL, TLS, Kerberos, IPSec, SSH, and WS-Security. In summary, they are similar in goal but different in implementation issues. This is the motivation to handle the modeling of security patterns by the following abstraction. As an example, on the domain-specific level we use the TLS mechanism (Dierks and Rescorla, 2008) as a concrete implementation of the SCP.

The TLS mechanism is composed of two phases: the *TLS Handshake* that establishes a secure channel, and the *TLS Record* in which this channel can be used to exchange data securely. The client initiates the TLS handshake by providing the server with a random number and information regarding the cryptographic algorithms it can handle. The server replies by choosing the actual algorithm to use, optionally requiring the client to authenticate itself, and by sending a random number of its own and its certificate issued by some Certification Authority trusted by both the server and the client.

To authenticate itself, in the final handshake message, the client includes its own certificate, a signature on all three handshake messages generated with the client's private key, and a third random number that is encrypted using the server's public key contained in the server's certificate. After verifying the certificates and signature, both client and server use the exchanged random numbers to generate session keys for generating and verifying message authentication codes (MACs) and for encrypting and decrypting messages during the TLS record phase.

Since the key used by the client to generate a MAC / encrypt a message is used by the server only for MAC verification / decryption and vice versa, and since these keys are based on one random number that is confidential to the client and the server, the keys establish a channel that provides authenticity and confidentiality for both client and server.

4.2.2. *Abstract syntax*

We propose an abstract syntax (a metamodel) by means of an OMG-style metamodel to construct the SEPM modeling language. The abstract syntax is based on previous requirements and describes various concerns, such as engineering, reuse and integration aspects. In our vision, we build on a semi-component pattern representation. Therefore, a security pattern is a subsystem that exposes pattern functionalities by interfaces and target security properties to enforce the security system requirements. Interfaces are employed to exhibit a pattern's functionality and manage its application. In addition, interfaces support interactions between security primitives and protocols within a specific application domain. The principal classes of the system and software engineering pattern metamodel (SEPM) are described with Ecore notations in Fig. 6 (not all classes and attributes are shown on the diagram to avoid cluttering). Their meanings are explained in the following paragraphs.

- SepmPattern. This block represents a security pattern as a subsystem that describes a solution for a recurring security design problem arising in a specific design context. A SepmPattern defines its behavior in terms of provided and required interfaces. Larger pieces of a system's functionality may be assembled by reusing patterns as components of an encompassing pattern or an assembly of patterns;

[6] http://eclipse.org/papyrus/.

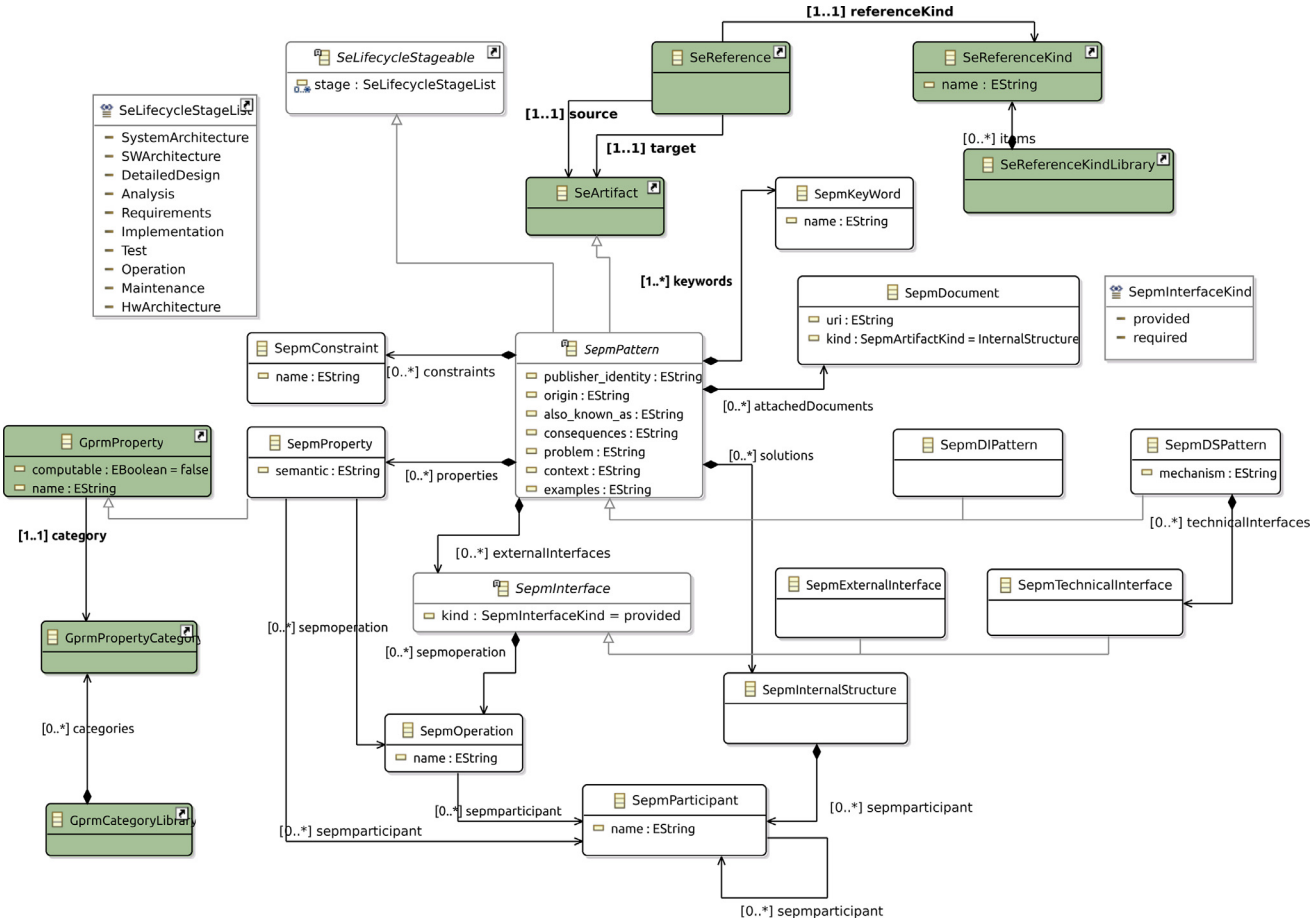

Fig. 6 – An overview of the SEPM.

the required and provided interfaces are wired together. A SepmPattern may be manifested by one or more artifacts.

- SepmDIPattern. This is a SepmPattern that denotes an abstract representation of a security pattern at the domain-independent level. This is the key entry artifact to model patterns at the domain-independent level (DIPM).
- SepmInterface. A SepmPattern interacts with its environment via SepmInterfaces, which are composed of operations. A SepmOperation represents the functional interface of the pattern. A SepmPattern represents the provided and required interfaces. A provided interface highlights the services exposed to the environment. A required interface corresponds to services required by the pattern to function properly.

  We consider two interface types:
  - SepmExternalInterface. This enables the implementation of interactions to integrate a pattern into an application model or to compose patterns.
  - SepmTechnicalInterface. This enables the implementation of interactions with security primitives and protocols, such as encryption, and specialization for specific underlying software and/or hardware platforms during the deployment activity. A SepmDIPattern does not have SepmTechnicalInterfaces.

  For our example, one may identify the following external interfaces:
  - *send*(*P*, *Q*, *ch*(*P*, *Q*), *m*),
  - *receive*(*P*, *Q*, *ch*(*P*, *Q*), *m*),

  with *P* and *Q* denoting the application sender *S* and application receiver *R*, respectively, *ch*(*P*, *Q*) their communication channel, and *m* a message.
- SeReference. This link is used to specify the relationship between patterns with regard to the domain and software lifecycle stage in the form of a pattern language. For example, a pattern at a certain software lifecycle stage *uses* another pattern at the same or different software lifecycle stage. SeReferenceKind contains examples of these links.
- 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 process levels.
- SeLifecycleStage. A SeLifecycleStage defines the development lifecycle stage in which the artifact is used. In our study, we focus on security pattern models. In this context, we use the pattern classification of Riehle and Züllighoven (1996), Buschmann et al. (2007, 1996).

- SepmProperty. This is a particular characteristic of a pattern related to the concern the pattern is addressing and dedicated to capturing its intent in a certain manner. The concept is employed to describe the security aspects of the subsystem to enforce the security system requirements. An example is security properties. The concept is used to describe the security aspects of the subsystem to enforce the security system requirements. A SepmProperty is defined through a name, a semantic, an expression and a category. The security attributes from (Avizienis et al., 2004) are categories of security properties. Each property of a pattern is validated at the time of the pattern validating process, and the assumptions are compiled as a set of constraints that must be satisfied by the domain application (Hamid et al., 2016). For our example, we define the following security properties:
  - *authenticity of action a and action b for an application entity P*. Thus, each time action *b* occurs, it must be authentic for an application entity that action *a* has occurred as well. For example, each time an application receiver *R* receives data *d*, it must be authentic for him that these are the same data *d* as the data sent by a specific application sender *S*, i.e., the action of sending these data, performed by a specific application sender *S*, must be authentic for the application receiver *R*.
  - *integrity of data*. When data *d* are received by the application receiver *R*, those same data *d* are sent out by the application sender *S*.
  - *confidentiality of data*. It denotes that only specific system entities are enabled to know the value of data *d*. For example, the private key is confidential to its owner and the application sender *S* trusts the confidentiality of the certificate authorities' (CA) private key.
- SepmConstraint. This is a set of requisites of the pattern. If the constraints are not met, the pattern is not able to deliver its properties. A constraint is a condition that holds or must hold during the application of a pattern. It is based on the notion of pre and post condition specification as commonly used in many formal methods. In our context, the assumptions derived during the formalization and verification processes of the pattern will be compiled as a set of constraints, for instance, resource constraints that will have to be satisfied by the domain application before the pattern application can be performed and after the pattern is applied. For our example, we specify constraints on the cryptographic algorithms, on the size of the cryptographic key, on the use of a security library and on the resources consumed by a security library.
- SepmInternalStructure. This constitutes the implementation of the solution proposed by the pattern: how the participants collaborate to carry out their responsibilities for the realization of the solution. Thus, the InternalStructure can be considered as a white box that exposes the details of the SepmDIPattern and the SepmDSPattern. One pattern can have several possible implementations, providing support for pattern variability.
- SepmParticipant. A listing of the components used in the pattern and their responsibilities in the design. In our context, a participant is a component type with a security-specific purpose. It's role is to add new functionality to the system that is specific to a security requirement the system should uphold.
- SepmDSPattern. This is a refinement of SepmDIPattern. It is used to build a pattern at the domain-specific level (DSPM). Because most known techniques that address security ability are cryptography-based models, we introduce the SepmDSPattern with a mechanism attribute to capture such notions in the SepmDIPattern model. In the example introduced in Section 4.2.1, TLS is one technique to achieve secure communication, and there are alternative ways to achieve the same goal. Furthermore, a SepmDSPattern has *Technical Interfaces* to interact with the platform. This is the key entry artifact to model the pattern at the DSPM.

4.2.3. *Concrete syntax*

To create model instances of the proposed metamodels, we must provide concrete syntaxes. In our context, we use a mixed syntax that combines structured-tree syntax, textual syntax and a UML-based diagrammatic syntax to describe the SEPM's concrete syntax. The basic idea is that the former defines problems and objectives, the textual defines properties and constraints, and the diagrammatic part defines roles and solutions (SepmInternalStructure). The objective behind this separation is that a solution defined in the pattern can be integrated (without losing information) into the application architecture only if both are specified in the same modeling language, e.g., UML. Conversely, the problem statement and objectives are independent of the chosen modeling language. This separation enables solutions defined in different modeling languages to share the same problem definition, which is useful because we are storing design patterns in a common repository, whereas model specifications in the structured-tree syntax are separately managed and stored. A pattern might eventually have multiple solutions defined in different modeling languages. The pattern discovery approaches, i.e., the mechanisms to browse or search patterns within the repository, are based on the non-diagrammatic part. Therefore, from a DSML construction perspective, design patterns are composed of two parts:

- *Structured-tree component*. Pattern definition that defines pattern properties and attributes, such as safety properties, resource constraints, development phases, and relationships. These data are used to ease pattern search and analysis.
- *UML-based diagrammatic component*. Pattern internal structure design files generated via additional tools, e.g., Rhapsody or Papyrus (UML editors), which are stored as XMI files and can be attached to the pattern description file (SepmDocument).

4.2.4. *Pattern verification process*

Formal modeling of patterns, combined with model checking, can prove some of the properties of a pattern's solution. We have made some experimentations (Hamid et al., 2016; 2011) to apply the techniques for formally proving security properties of security patterns provided by the security modeling framework (SeMF) developed by Fraunhofer SIT(Gürgens et al., 2005), following the two abstraction levels of the pattern representations. In contrast to other formal security engineering

methods (Devanbu et al., 2000; Landwehr, 1981; Paulson, 1996), the used formal security framework, referred to as SeMF, is not following the attack nor the risk based approaches. Its basis are a set of desired security properties and associated assumptions. With SeMF it is possible to validate if properties like trust, authenticity or confidentiality hold under given assumptions. The side benefit is case a stated assumption does not hold is that possible consequences in regard to security properties can be estimated. The proof itself is conducted mostly with pencil and paper and the resulted proof artifacts will be utilized by the designer as input to the pattern-based development process. For more information regarding the formal framework and the definitions of security properties we refer the reader to Fuchs et al. (2010) and Hamid et al. (2016).

Here, we report on an experiment to apply SeMF to the DIPM and DSPM for the secure communication pattern targeting the security property *each time the server side of the communication channel receives a message on the channel, for the server it authentically originates from the client side of the channel.*

4.3. *Step 3: Definition of security patterns*

Once we have developed the DSML's concrete syntax in Step 2, we can create the set of security patterns to share the security expertise within the domain of interest. During this step, the patterns are constructed such that they conform to the metamodel description adopted in Step 2. To foster technology reuse across domains, the patterns are stored in a repository, such as that described in Hamid (2017), thus reducing the amount of effort and time needed to design a complex system.

The target representation is the domain-independent level (DIPM), while still conforming to the SEPM metamodel. At the DIPM level, this description reveals the following elements: *interfaces* of type SepmExternalInterface, *security properties* of type SepmProperty and *solutions* of type SepmInternalStructure. Moreover, for classification and relationship definition purposes, additional information may be defined, e.g., *lifecycle stages* of type SeLifecycleStage and *relationships* of type SeReference, respectively.

The first task is to create a basic pattern subsystem as an instance of the SepmPattern. The instance is given a name and a set of attributes that correspond to the pattern. The description, with varying levels of abstraction, is managed by inheritance. Once the basic pattern subsystem is specified, interfaces are added to expose some of the pattern's functionalities. For each interface, an instance of *SepmExternalInterface* is added to the pattern's interface collection. The next step after creating interfaces is the creation of property instances. An instance is created in the pattern's property collection to specify every identified security property. A property is given a name and an expression based on external interfaces in a property language.

We continue our illustration using the example of the *secure communication pattern*. For the sake of simplicity, we specify only those elements related to both Step 2 and Step 3 that are required to explain our approach. The secure communication pattern enables a secure data exchange between components over a non-secure communication channel (e.g., Ethernet), thus ensuring the integrity and confidentiality of the data and the authenticity of the sender. Messages sent among distributed system functions (components) shall arrive from authorized sender(s) without data modification. If any attacker sends a message or modifies an existing one, this message should be discarded by the receiver security communication layer, and the destination function might be informed of this action.

The domain-independent pattern uses an abstract channel and provides message authenticity under specific assumptions, particularly with regard to the trust of the server with the precedence of its own receive action via a send action performed by the respective client. The channel authentication layer uses a proprietary key to authenticate itself in the communication. On the receiving side, the channel authentication performs necessary checks to ensure that the received message is authentic and that the data receiving function trusts the channel authentication. In our example, we identified two external interfaces, one for the client and one for the server, providing the following functions:

- *establCh*(P, $ch_k(P, Q)$): P establishes a channel ch_k with Q,
- *send*(P, $ch_k(P, Q)$, m): P sends message m on the channel ch_k shared with Q,
- *recv*(P, Q, $ch_k(P, Q)$, m): P receives and accepts message m on the channel ch_k shared with Q,
- *closeCh*(P, $ch_k(P, Q)$): P closes the channel ch_k shared with Q.

with $P \in \{C_1, \ldots, c_n\}$ and $Q = S$ or vice versa, $ch_k(C, S) = ch_k(S, C)$ and $k \in \mathbb{N}$.

The next concern of the process is the definition of the pattern security properties. The supporting activities require the availability of a set of property libraries. For the example of the secure communication pattern at the DIPM level, we specify the security property: "authenticity of receiving for the server". For this we note that the server sees its own actions, thus the action of receiving a message is particularly authentic for the server itself. Further, the server will never accept a message on a channel it has already closed: Every receive action for a particular message m occurs within an active channel. This means that receipt and acceptance of a message by the server occur authentically for the server within the phase that corresponds to the active channel $ch_k(P, Q)$ being established and closed, respectively. To type the category of this property we use a category from the ones defined in the security category library "Authenticity". For the other types of properties, we have to consider a concrete solution. There exist various different possibilities for such patterns that refine the abstract communication channel and thus the abstract pattern. One possibility is, e.g., to execute a Diffie Hellman based key exchange algorithm. Another possibility that we will focus on in the next section is the establishment of a TLS channel.

Furthermore, each pattern is studied to identify its relationships with other patterns belonging to the same application domain based on the engineering process activity in which it is utilized. The purpose of this activity is to organize patterns into a set of pattern systems. Moreover, this step should include all activities that support pattern producers in managing the relationships among these patterns, which can be defined in pattern relationship model libraries. At each stage (phase) n of the system engineering development process, the patterns identified in the previous stage (phase) $n-1$

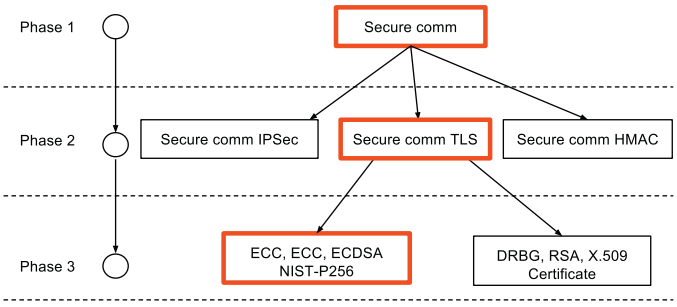

Fig. 7 – Tree-shaped pattern refinement and specialization.

can assist in the selection process during the current phase. Similarly, we specify model libraries for patterns classification. At each stage of the system engineering development process, the appropriate patterns are identified via a classification process.

After an initial analysis of the various artifact sources, including standards and existing applications, the designer determines the stage of the engineering process lifecycle (system concept, system architecture, software architecture, and detailed module design) in which each pattern can be defined; moreover, whether the pattern is domain-independent or domain-specific can be determined. For this purpose, we select the pattern classification of Riehle and Züllighoven (1996), Buschmann et al. (2007, 1996), who defined *system patterns, architectural patterns, design patterns* and *implementation patterns* to create the *SeLifecycleStage* model library. In addition, a pattern may be linked with other patterns and associated with property models using a predefined set of reference types, at a very high-level (Noble, 1998) or including details regarding what part of a pattern is used, refined, or combined (Hauge, 2014). Here, we create the *SeReferenceKind* model library to support the specification of relationships across artifacts (e.g., *refines, specializes* and *uses*) as an extension of the relationship classification proposed in (Noble, 1998).

In the context of our work, certain patterns have a meaningful representation at the system level, at which general system blocks are defined and domain concepts are expressed (e.g., system skeleton). However, their representations might not be directly refined in later phases because they represent concepts that are meaningful at only the architectural level. In contrast, other patterns might be meaningful only in later design phases as indirect specializations of an architectural concept, e.g., a secure remote read out (SRR) software pattern is a specialization of an architectural skeleton pattern. In addition, the same pattern may have multiple instantiations and specializations in each phase (e.g., a Wakeup Service is linked to a hardware component). Therefore, as shown in Fig. 7, a given design pattern (secure communication pattern) in the repository might follow a tree-shaped refinement and specialization flow, representing different lifecycle phases, different refinements and specializations, and new pattern representations in later phases. For instance, TLS provides the mechanisms for confidential information exchange through ECC (NIST-P256) to support secure channel definition and content data encryption and the authentication of communication partners through digital signatures using ECDSA (NIST-P256).

4.4. *Step 4: Definition of security patterns for a specific domain*

At the domain-specific level (DSPM), the security pattern and some of its related elements are also created by inheritance. Once a SepmDSPattern is created, every pattern external interface is identified and modeled as a refinement of the DIPM's SepmExternalInterface in the pattern's interfaces collection. Then, following the pattern's description of the particular solution that is represented, each of the pattern's technical interfaces is identified and modeled by an instance of SepmTechnicalInterface in the pattern's interfaces collection.

In the context of our experiment, the metrology domain-specific pattern must be compliant not only with the generic security standards but also with metrology-specific security standards. Some patterns, for example Wakeup Service are defined exclusively for the metrology domain and others are adapted from generic security standards. For instance, in the Common Criteria Protection Profile of the smart meter gateway (BSI, 2014), the use of the TLS protocol (refer to Section 4.2.1) providing a secure communication channel between two communication partners is marked as mandatory for all connections between a gateway and wide area network. Therefore, the domain specific *secure communication pattern* as a refinement of the domain independent one is based on TLS. It implements the abstract channel by establishing a TLS channel based on shared secrets. The TLS channel in turn provides the server's trust into precedence of message receipt via sending of messages performed by the respective client, again under certain assumptions. The TLS pattern uses the server's public and private keys for RSA encryption and decryption (which is part of the pattern itself and does not use a further RSA pattern) and an additional pattern such as DRBG for random number generation that must satisfy specific properties. A client and a server establish a communication channel by exchanging three random numbers (two generated by the client and one generated by the server) and generating a set of different session keys based on these random numbers. Because the client's second random number is sent to the server encrypted with the server's public RSA key, this random number, if generated confidentially, stays confidential, and hence all session keys are confidential, thus establishing an authentic and confidential channel for client and server.

For instance, when using TLS as a mechanism related to the application domain to refine the secure communication

pattern at DSPM, we manage the following artifacts. The external interfaces are defined as a refinement of the DIPM external interfaces. In addition to the refinement of the concepts used at DIPM, the process involves the definition of technical interfaces. These two activities are complementary and can be mutually reinforcing when undertaken simultaneously. For instance, the establishment of a channel using the TLS mechanism is a refinement of the DIPM action *establCh*(*P, ch*(*P, Q*)): We add the random numbers generated by *P* and *Q* using the corresponding technical interfaces (*genRnd*(*P, Q*)), which results into *establCh*(*P, ch*(*P*, rnd_P, $preMS_P$, *Q*, rnd_Q)). A subset of the functions provided by the external interfaces is:

- *establCh*(*P, ch*(*P, rnd*$_P$, *preMS*$_P$, *Q, rnd*$_Q$)): *P* establishes a channel *ch*(...) with *Q*,
- *send*(*P*, *ch*(*P*, rnd_P, $preMS_P$, *Q*, rnd_Q), *m*, *mac*($preMS_P, \ldots, m$)): *P* sends *m* and the corresponding MAC to *Q* on the channel *ch*(...),
- *recv*(*P*, *ch*(*P*, rnd_P, $preMS_P$, *Q*, rnd_Q), *m*, *mac*($preMS_P, \ldots, m$)): *P* receives *m* and the corresponding MAC from *Q* on the channel *ch*(...),
- *closeCh*(*P, ch*(*P, rnd*$_P$, *preMS*$_P$, *Q, rnd*$_Q$)): *P* closes the channel *ch*(...) shared with *Q*.

with *P, Q* as defined above, rnd_P and $preMS_P$ denoting a random number and the premaster secret, respectively, generated by *P*, *m* denoting a message and *mac*($preMS_P, \ldots, m$) the message authentication code generated using the premaster secret.

The technical interfaces are defined as a set of functions related to the use of TLS to refine the secure communication pattern. A subset of the functions provided by the internal interfaces is

- *genRnd*(*P*, rnd_P): *P* generates a random number rnd_P,
- *getCert*(*P, cert*): *P* has access to its certificate,
- *getKey*(*P*, PuK_{CA}): *P* has access to the CA's public key,
- *verifyCert*(*P*, PuK_{CA}, *cert*): *P* verifies the certificate *cert*,
- *genMac*(*P*, *ch*(*P*, rnd_P, $preMS_P$, *Q*, rnd_Q), *m*, *mac*($preMS_P, \ldots, m$)): *P* generates the message authentication code (MAC) for a message using its own TLS shared secret for MAC generation

At the DSPM level, we define several concrete security properties, including "authenticity of sending and receiving for the server." Other properties related to the usage of TLS may also be defined, such as "confidentiality of the session key."

4.5. *Step 5: Adaptation for a specific domain development environment*

The final steps (5 and 6) are performed to support the development of a specific system. Step 5 identifies appropriate patterns and creates tailored versions that represent model concepts in the domain of interest and that can be adapted to both the system development process and the development environment. The selection of a pattern is primarily the choice of the developer. There are various considerations that may narrow and simplify this choice. The first is the purpose of the pattern application. Although this purpose cannot be generally formalized, certain patterns address requirements that are defined by domain standards (e.g., security). If these requirements are stored in a model library and referenced in the definitions of the patterns, then the selection of patterns could be driven by the selection of (domain) requirements. The second consideration is that patterns can be classified with respect to several properties, one of which is the stage of the engineering process lifecycle discussed in Section 4.3 - a pattern may be relevant to the system, its architecture, or to aspects of its design or implementation. Thus, it must be possible to filter available modeling artifacts based on this classification.

In the context of our work, the target domain development environment is IBM Rational Rhapsody, and the descriptions of the model transformations are based on the QVT operational language. Therefore, the design of a given pattern can be regarded as a single package that contains one subpackage per lifecycle phase of the engineering process; each of these phases can contain design modules and additional sub-packages associated with particular specializations and refinements. Thus, imported patterns are stored inside a dedicated package that facilitates searching within the package tree of each design. Moreover, to foster reuse, the pattern artifacts related to that phase are instantiated from the repository to the vehicular modeling tool as a reference package. The right part of Fig. 9 shows a pattern design deployed in packages using the IBM Rational Rhapsody tool. Each pattern design package generally contains the following items:

- Any information that is required by the end-user pattern integrator, e.g., a UML class or SysML block, with interfaces that enable the interconnection of patterns with a given system design.
- Additional detailed information of interest, e.g., a "structure" package that contains the static internal structure (e.g., class diagram) and the dynamic structure (e.g., sequence diagram).

4.6. *Step 6: Reuse for a specific system development*

This section focuses on the use of patterns in a software development process. The integration of a pattern involves the application of a solution provided by that pattern in an existing application architecture to take advantage of its benefits. We cannot simply copy such a solution into the architecture under development. Instead, we must account for the interplay between elements that previously exist in the application and the elements of the pattern.

To address this issue, a specific activity called *Integration*, which was previously studied in Hamid et al. (2012), is used herein. In the context of our example, by executing a tailoring activity, the pattern is exported in an XMI file. Then, it must be imported from Rhapsody. As shown in the right part of Fig. 9, once the pattern is imported in Rhapsody as a package, a project tree is generated and its artifacts are available in the project. Therefore, in each phase, the system developer executes the search/select task on the repository to tailor appropriate patterns for the modeling environment using the identification and the tailoring processes described in Section 4.5. The developer then integrates them into the application models following an incremental process. In the software

architecture phase, our process flow can be summarized by the following steps:

1. The software architect searches for different specializations (at the software architecture-definition level) of the patterns to complement the design.
2. The software architect selects the appropriate set of identified patterns.
3. The software architect imports the software architecture design perspective of each pattern into the vehicular modeling environment (Rhapsody) as a reference package. The application developer is responsible for linking the pattern interfaces to integrate the pattern at that specific level.
4. The software architect integrates the pattern into the existing software architecture design diagrams.

Moreover, in our work we consider the definition of guidelines to correctly use security patterns. At security application design-time, formal modeling of patterns can prove some of the properties of a pattern solution. It must be ensured that the assumptions used to prove the correctness of the pattern are indeed satisfied by the particular environment of the application, following the unified modeling and formal design framework for pattern definition and verification processes proposed in Hamid et al. (2016). Stored in a repository, validated security patterns are then made available for integration into an MDE process to develop secure applications for various domains. Beyond this, we store in the repository associated verification artifacts, i.e., properties and assumptions.

When using the pattern, an application developer will be concerned with the security requirements expressed in terms of the external interface, i.e., in terms of the function calls used by the application. Thus, any formal proof needs to refer to these function calls. Alternately, the solution described in a respective DSPM pattern must be modeled in terms of refined function calls. Intuitively, we propose handling the assumptions that the proof is based on as a set of constraints that need to be satisfied in order for the pattern and the solution it specifies to provide the proven pattern properties in terms of its interfaces. In other words, we execute the different steps of the proof and the related assumptions as requirements on the external and technical interfaces. This is the basis of the correct integration of patterns

For example, when proving the authenticity property the TLS Handshake (Hamid et al., 2016), we used the following assumptions:

- Ass1. The client does not encrypt the premaster secret with two different public keys (it might encrypt it twice using the same key).
- Ass2. The receiver application entity must not accept a message after having closed the respective channel.
- Ass3. The shared secrets must be deployed in such a way that they are only known to the sender and the receiver application entities.
- Ass4. The receiver will not use the shared secret of the sender to compute HMACs.
- Ass5. The random number generator produces unpredictable random numbers.

We now introduce some constraints derived from theses assumptions that the application developer needs to verify to ensure that the pattern used indeed provides the required authenticity property:

- Implementation of sender and receiver (i.e. client and server) application entities. The sender application entity must be implemented adhering to assumption Ass.1. The receiver application entity must be implemented in compliance with assumption Ass.2.
- Key Handling. The HMAC algorithm works with shared secrets deployed according to Ass.3. Further, it must be ensured that the receiver satisfies Ass.4. This also means that the same shared secret must not be used for bi-directional transmissions.
- Random number generation. The sender application entity must ensure that it uses a random number as basis for the premaster secret and that the random number generator adhere to Ass.5.

5. Tool support

In this section, we propose an MDE tool chain to support the proposed approach and to assist the developers of model- and pattern-based secure software systems. As discussed below, the proposed tool chain is designed to support the proposed metamodels; hence, the tool chain and the remainder of the activities involved in the approach may be developed in parallel. The appropriate tools for supporting our approach must fulfill the following key requirements:

- Enable the creation of the UML class diagrams used to describe the pattern metamodel in our approach.
- Enable the creation of a concrete syntax.
- Support the implementation of a repository to store pattern models and the related model libraries for classification and relationships.
- Enable the creation of pattern models and the related model libraries and the publication of the results into the repository.
- Support the administration and the internal management of the repository.
- Enable the creation of visualizations of the repository to facilitate its access.
- Enable the creation of application models.
- Enable transformations of the models from the repository format into that of the target modeling environment.
- Enable the integration of application models and imported patterns.
- Support application-specific code generation.

To satisfy the above requirements, we develop an MDE tool chain on top of the current version of SemcomDT[7] (SEMCO model development tools) to support all the steps in our approach.

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

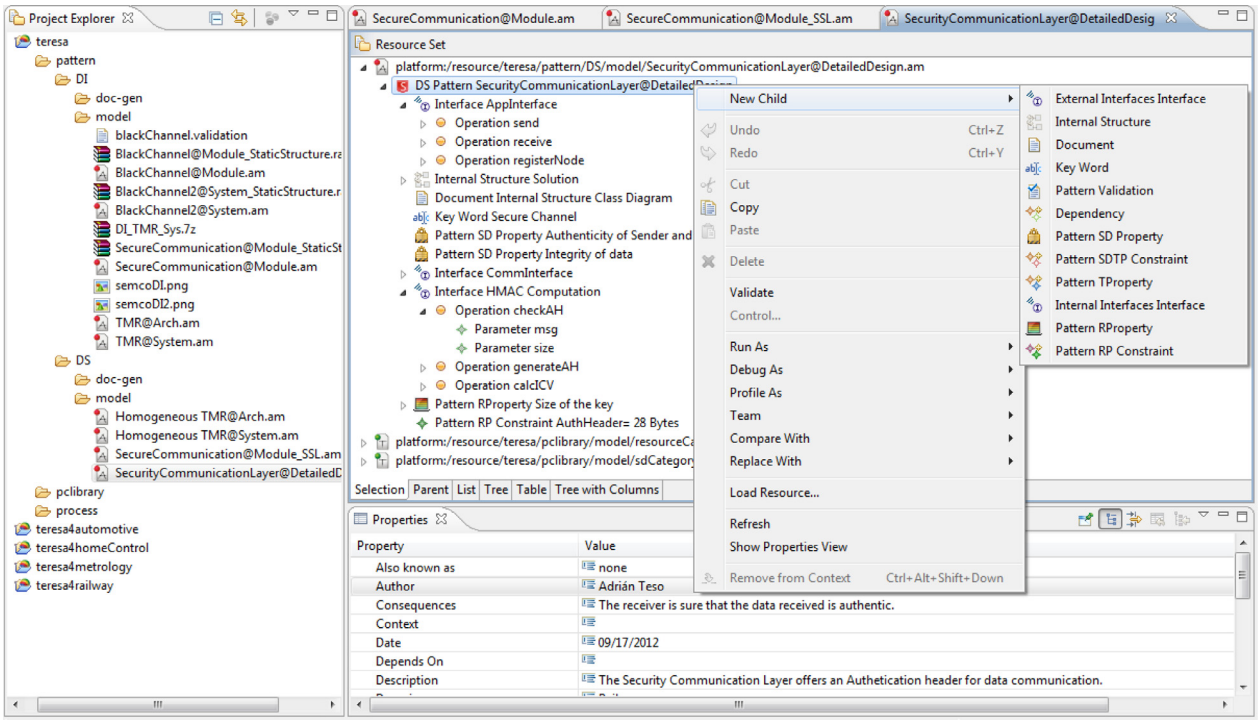

Fig. 8 – Designing a pattern.

The repository is implemented using the Eclipse CDO[8] framework. We use the same process flows for the design and implementation of a reuse model repository such as the one described in Hamid (2017). We apply metamodeling techniques that enable the specification of the repository structure and interfaces to content in the form of modeling artifacts and model transformation techniques for the purpose of generation. To populate the repository, we construct a pattern design tool (Arabion) to be used by a pattern designer. Arabion interacts with the repository for publication purposes. The pattern design environment is presented in Fig. 8. A design palette is shown on the right, a tree view of the project is shown on the left, and the main design environment is presented in the middle. Furthermore, Arabion includes mechanisms for verifying the conformity of the pattern with the SEPM metamodel and for publishing the results to the repository.

Rhapsody is used as the domain-specific design software tool to design (and implement) the system using UML/SysML modeling languages. For example, it can be used to design systems based on packages, where one package might contain design diagrams and/or additional packages. Based on this approach, the design of a given pattern can be considered a single package that contains one sub-package per safety engineering process lifecycle phase; each of these phases might contain design modules and additional sub-packages associated with specializations and refinements. Thus, the access tool provides the option to export patterns in a format that can be imported by the Rhapsody tools. Therefore, a customized access tool, such as the one shown in Fig. 9, is developed to construct connections between the metrology development environment and the repository of patterns. The access tool provides a set of functions to assist in the search, selection and sorting of patterns:

Rhapsody is used as the domain-specific design software tool to design (and implement) the system using UML/SysML modeling languages. For example, Rhapsody can be used to design systems based on packages, where one package might contain design diagrams and/or additional packages. Based on this approach, the design of a given pattern can be considered as a single package that contains one subpackage per security engineering process lifecycle phase; each of these phases might contain design modules and additional subpackages associated with specializations and refinements. Thus, the access tool provides the option to export patterns in a format that can be imported by the Rhapsody tools. Therefore, a customized access tool, such as the one shown in Fig. 9, is developed to construct connections between the metrology development environment and the repository of patterns. The access tool provides a set of functions to assist in the search, selection and sorting of patterns:

1. Keyword based search for patterns: Feedback from engineers of companies with a line of business in the metrology domain revealed a clear preference for a keyword-based search when querying the repository for patterns.
2. Context based search for patterns: In some cases, engineers need to find specializations or linked patterns for the solution provided by an already instantiated pattern.
3. Pattern browser for manual pattern search: The access tool provides a dedicated pattern browser tab for

[8] http://www.eclipse.org/cdo/.

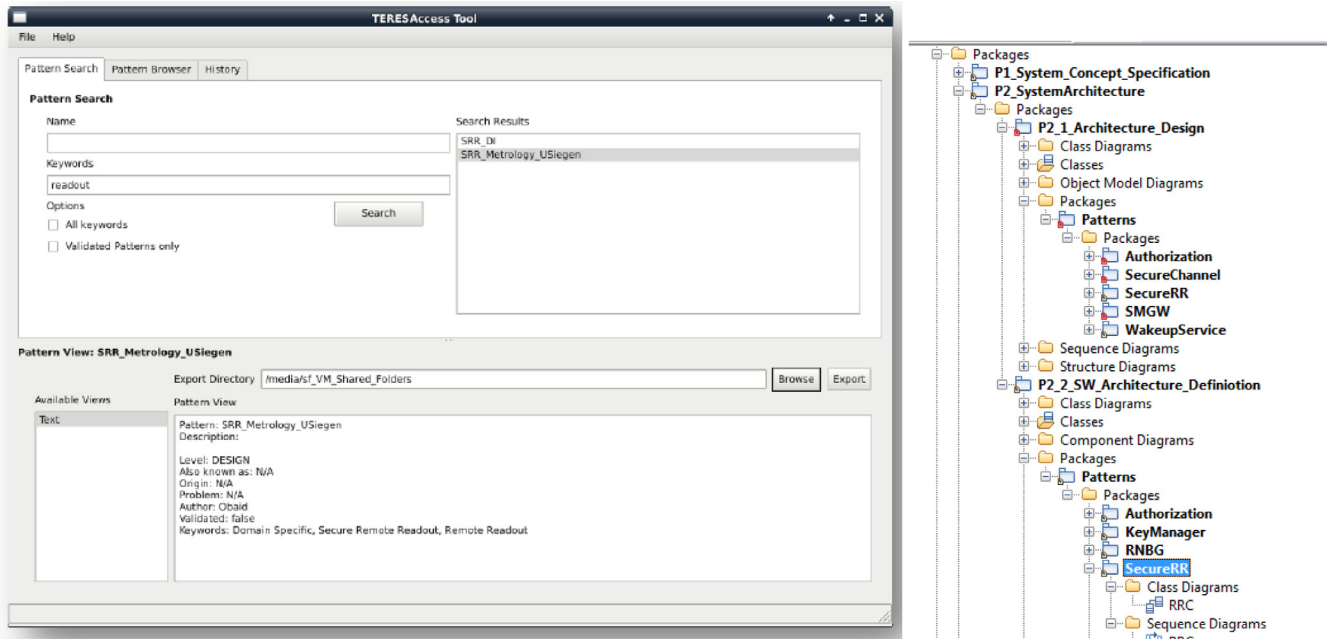

Fig. 9 – Metrology access tool.

manually browsing the pattern repository for new patterns or for obtaining an overview of the solutions available.

4. History of already exported and instantiated patterns (project related): To ensure consistency, it is important to know which patterns have already been instantiated and integrated during the different phases of the development process.

For example, as shown in the left part of Fig. 9, the tool assists in selecting appropriate patterns through key word searches and lifecycle stage searches. The results are displayed in the search result tree as system, architecture, design and implementation patterns. When a pattern is selected, the access tool instantiates the pattern in the domain-specific tool, as shown in the right part of Fig. 9. Because this task is performed during product development, the selected pattern must be compliant with the current phase of the domain process and with the user tools. By accessing the repository, we introduce features based on model transformation techniques to adapt the pattern model to the target development environment. In our work, the target format is a subset of UML that can be imported using Rhapsody and the model transformations, which was developed using the Eclipse implementation of QVTO.

As shown in Fig. 9, the system architect types in a query and searches the readout to find the most suitable design pattern instantiation for the corresponding phase. The system architect analyzes all possible options, selects “SRR_ Metrology” and clicks *Export*. By clicking the *Export* button, the pattern is exported as a Rhapsody-referenced package to the selected directory. Once the pattern is imported into Rhapsody as a package, a project tree is generated, and its artifacts are available for use in the project.

6. Evaluation

In this section, we first report an industrial case study performed in the metrology domain (Section 6.1), followed by a description of a survey performed among metrology domain experts to better understand their perceptions of our approach (Section 6.2). The case study enables us to determine whether the pattern-based approach leads to a reduced number or to a simplification of the engineering process steps, whereas the survey assists in assessing whether domain experts agree regarding the benefit of adopting the pattern-based approach in a real industrial context.

6.1. *Case study*

We use smart meter gateway systems to exemplify the proposed approach. A smart meter gateway (BSI, 2014), is a system capable of connecting to several meters for different commodities, such as electricity, gas, water or heat, and communicating with households and other remote entities, such as regulations. It enables additional services, such as accurate monthly bills and consumption information regarding the actual time of use. Smart meters utilize embedded systems providing network connectivity to the backend. In this scenario, communicating different information (e.g., readouts, consumption data, parameters) with several entities raises additional security concerns. It is important to have authentic information about the energy consumed at different measurement points. Hence, security for metering has become an im-

portant question, and security functionality has been shifted completely to the gateway. To address these security concerns, several design techniques from related standards and other techniques, such as digital signatures to ensure the authenticity and integrity of measurements, have been introduced to the metrology domain.

In this subsection, the adaptation of metrology processes to incorporate the pattern-based approach is described. We evaluate the proposed approach in the construction of a methodology that is adapted for engineering secure systems by combining the MDE process and security patterns. Furthermore, we evaluate the usefulness of the patterns with respect to increasing engineering productivity. The goal of the case study was to determine the feasibility of the approach and the level of effort involved in its application for this case study. In the presentation of the case study and its results, we do not show the complete resulting model because it contains proprietary information from our industrial partner.

6.1.1. *Nature of the case study*

The aim of this case study is providing a methodology to improve existing approaches to engineering secure systems using MDE. Therefore, the case study can be seen as an improvement case study (Runeson and Höst, 2009). To obtain scientifically sound results that enable comparison with current practice, it is necessary to perform a study in which comparable systems are designed in parallel by equivalent engineering teams. This method is not feasible within the scope of our work because of a lack of resources required to develop the system addressed in the case study twice. Note, however, that the stakeholders' perceptions discussed in Section 6.2 could be used for comparison.

The *smart meter gateway* demonstrator, which is a simplified version of a real gateway PP that enables connecting to several meters for different commodities, such as electricity, gas, water or heat, and communicating data with remote entities in a WAN (Wide Area Network) or HAN (Home Area Network), is visualized in Fig. 10. Additionally, for confidentiality reasons, we use a small but realistic setting to illustrate the security pattern-based approach proposed herein. The meter in the LMN (Local Metrological Network) is an electricity meter that is located in the same housing as the gateway. The combined gateway/meter housing is sealed and defined as a secure environment. The electricity meter communicates measurement information with the gateway.

The main function of this demonstrator is to ensure that the measurement information is processed in the gateway and exchanged with the remote readout center in the WAN. The remote readout center (RRC) functions as an authorized external entity, as depicted in Fig. 10. The gateway acts as the connecting piece between three different networks: WAN, LMN and HAN. For consumer interaction, the gateway provides a display and several LEDs for status indication. The display can be seen as part of the home area network. Furthermore, a hardware security module (HSM) is part of the gateway. This is required by the protection profile to provide hardware cryptographic functionality and a secure storage for the gateway system. More detailed information regarding the scenario and the roles involved is given in Gonzalez and Weber (2013).

In our context, the term smart meter gateway is a synonym for the communication unit of modern smart meters. In contrast to the communication units of classical meters, gateways offer a wide range of new functions needed for a secure smart grid operation. Most of their functionalities address the processing and exchange of information between the different actors and networks. Smart meters and their gateways generate security- and safety-relevant, as well as privacy-sensitive, data and transmit them over possibly insecure networks. A special kind of protection is required for these data. Among other approaches, the German Federal Office for Information Security issued a common criteria protection profile (PP) for the communication unit of a smart meter (BSI, 2014). The security requirements of the proposed case study are extracted from a simplified analysis of the PP standard with the consideration that the objective of the case study is not the development of complete, certifiable and interoperable metrology system but the provision of a case study that is as realistic as possible and that can be developed within the scope our study.

6.1.2. *Research questions*

The purpose of this study was to address the following two research questions:

- RQ1. *Is the approach feasible?* More precisely, this question concerns (1) whether the developed conceptual model allows capturing security patterns, and (2) whether the proposed methodology, modeling language and tool suite allow the engineering of secure systems using patterns based upon a security pattern's conceptual model. For answering **RQ1**, we do not consider the creation of the domain models and the development of a reuse mode repository required in our approach. These activities are technically realizable and do not require a feasibility investigation (Hamid, 2014).
- RQ2. *Is the effort involved in the application of the approach acceptable?* The ultimated goal is the adoption of the approach in industry. Effort required in the usage of a new approach is an important factor for its successful adoption. For **RQ2**, we measure the effort spent throughout the case study, including the development of the methodology and tool suite for PBSE. The goal is to assess whether experts find the level of effort reasonable.

6.1.3. *Description of the application*

Fig. 11 provides a diagram of the selected cases of the developed smart meter system, illustrating their relationships and identifying them as either security-relevant or non-security-relevant cases. These cases can be described as follows

1. Exchange measurement data and information on actual consumption and generation of electricity: (1) Gateway establishes a remote connection with the authorized external entity; (2) gateway sends measurement information; and (3) gateway closes the remote connection with the authorized external entity.
2. Exchange status data and parameters and administrate the device.
3. Exchange measurement data on consumed and generated commodities: gateway sends command to read

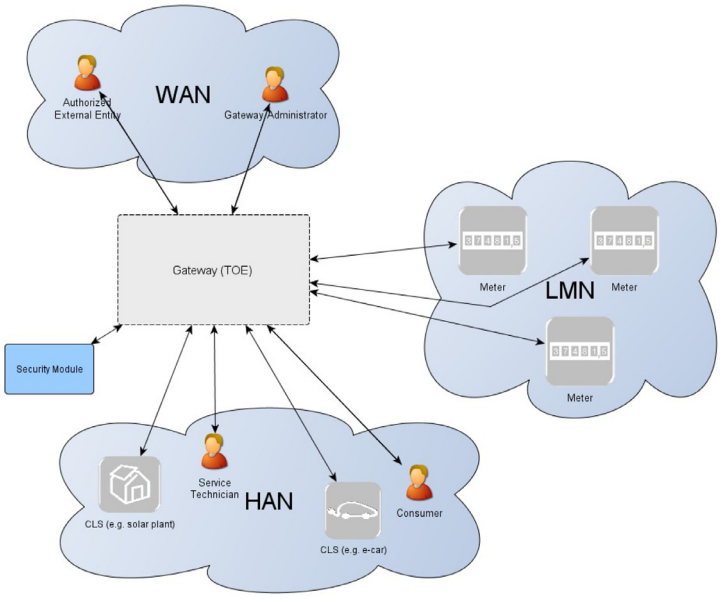

Fig. 10 – Smart meter system diagram.

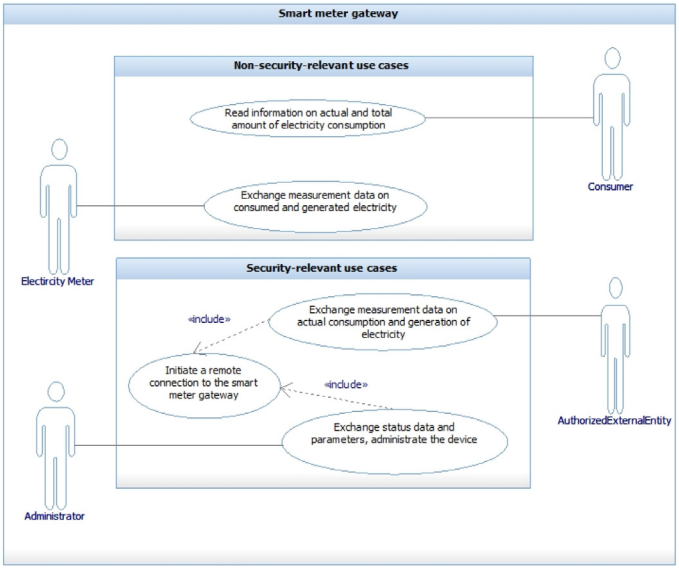

Fig. 11 – Part of a smart meter use-case diagram.

data from the meter; (2) meter sends requested data to the gateway; and (3) gateway processes and stores data.

4. Read information on actual electricity consumption and total amount of energy consumed: (1) acquire measurement and (2) display measurement on the LCD.

Furthermore, a direct connection between the smart meter gateway and the remote readout center is assumed. In practice, different communication and networking topologies between a gateway and authorized external entities in a WAN are deployed. Some of them use data concentrators or substations in their nodes. According to the protection profile and its intended end-to-end security, the topology is invisible for the gateway and will not influence the described scenario.

The security characteristics of the smart meter gateway can be easily extracted from the protection profile of the gateway of a smart metering system provided by the Federal Office of Information Security (BSI, 2014). The requirements described below are a summary (with some revisited information) of the security requirements and are defined as requirements to protect the corresponding assets:

1. *Meter data*. Meter readings that allow calculation of the quantity of a commodity, e.g., electricity consumed over a period.

(a) Integrity and authenticity (comparable to the classical meter and its security requirements)
(b) Confidentiality (due to privacy concerns)

2. *Consumption data*. Billing-relevant part of meter data.
   (a) Integrity and authenticity (comparable to the classical meter and its security requirements)
   (b) Confidentiality (due to privacy concerns)
3. *Status data*. Grid status data, subset of meter data that is not billing-relevant.
   (a) Integrity and authenticity
   (b) Confidentiality (due to privacy concerns)

6.1.4. *Data collection procedure*

The procedure used for developing these demonstrators closely followed the approach described in Section 4. Given the domain requirements, a conceptual model of security patterns was built by studying the state-of-the-art with respect to pattern modeling and formalization and by analyzing standards. This work was done by the two authors. The model was presented to a group of 8 security experts during a TERESA advisory board meeting workshop. During this workshop, the concepts were presented and explained in detail. Taking into account some of the feedback, the model was subsequently revised. The next step was the creation of the modeling language (abstract and concrete syntaxes) for the specification of security patterns. We proposed an abstract syntax (a metamodel) by means of an OMG-style metamodel. The abstract syntax is based on the conceptual model of security patterns. We then reviewed and refined this DSML over several meetings with an expert in security engineering at the partner company where we were conducting the case study. This work was done by the first author. Then, we proceeded with the implementation of the MDE tool chain, including the development of a customized access tool (see Section 5). This work was performed by the first author in close collaboration with two PhD students.

The third and fourth steps involved the definition of security patterns for populating the repository. This included the (one-time) effort made to identify and understand security patterns and the relationships between them, reading each pattern's standard documentation and creating the pattern models using the MDE tool chain. The resulting patterns are specialized with respect to the needs of the supplier and include concepts specific to the smart meter gateway systems developed by the supplier. Steps 3 and 4 were carried out by the second author, and the results were reviewed by the experts at the partner company.

The next steps involved the development of an application using patterns. To this end, we first created a generic domain model (i.e., smart meter gateway system model) using a description found in BSI (2014). Once the domain model had been created, the pattern-based process was carried out. As mentioned earlier in Section 4, the integration phase of our approach requires finding and tailoring suitable patterns to a form that is appropriate for the targeted development environment using the MDE tool chain. Steps 5 and 6 were carried out by the two authors, and the results were reviewed by the experts at the partner company.

Table 2 – List of patterns.

Pattern	Origin
Authorization (AZ)	(BSI, 2014; Fernandez, 2013)
Secure Communication (SC)	(Dierks and Rescorla, 2008; Rescorla and Modadugu, 2012; Schumacher et al., 2005)
Key Manager (KM)	(Callas et al. 2007; Schumacher et al., 2005)
Random Number Generator (RNG)	(Barker and Kelsey, 2012)

6.1.5. *Results*

Here, we present the results of our case study. Because elements of this study were discussed in Sections 4 and 5 to explain our approach, we provide only an overview of the outcomes of the case study (Step 3, Step 4, Step 5 and Step 6).

6.1.5.1. *Definition of security patterns (Step 3)* In this step, the architects analyze the system security requirements and identify possible security patterns to be used. Table 2 presents the list of patterns to be used in the metrology demonstrator. This list populates the repository of security patterns for the metrology domain through the MDE tool set presented in Section 5.

6.1.5.2. *Definition of security patterns for a specific domain (Step 4)* The domain-specific perspective of a pattern is dependent on the solution/product; each (commercial) implementation provides different characteristics and features. In the context of our work, a metrology-specific model is constructed based on previous patterns using the tool suite. We use the same process flows as applied for the domain-independent representation, although the appropriate features of the toolset are used to create and deposit the corresponding domain-specific representations of these patterns into the repository. Here, we present a subset while focusing on the specific metrology realizations.

- *Smart Meter Gateway (SMGW)*. The protection profile for the smart meter gateway requires an absolute separation of the communication interfaces at the logical as well as physical level. For an application engineer, it is very important to pay attention to this requirement in an early high-level design phase to avoid problems. The smart meter gateway pattern provides a component diagram, as depicted in Fig. 12, to integrate the separated interfaces at the architecture level. An alternative to the BSI proposal for the architecture of the interconnection of smart metering devices, as shown in Fig. 10, was proposed by the Department of Energy and Climate Change in the UK (DECC, 2015). The "Communications Hub" in both architectures are similar. The only difference is that the HAN includes the LMN as well as the end user devices.
- *Secure Remote Readout (SecureRR)*. This pattern is defined exclusively for the metrology. According to the PP (BSI, 2014), the data to be remotely read out (measurement data and compulsory relevant log for meters) is protected against manipulation with the help of digital signatures.

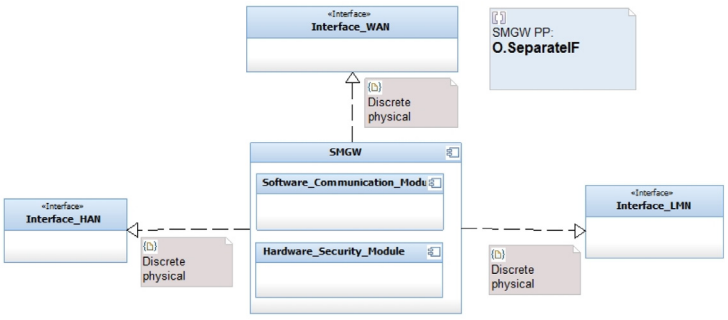

Fig. 12 – Component diagram of the gateway's WAN, HAN and LMN interfaces.

At the moment the measurement data becomes available, a timestamp is appended, and a digital signature is created over the data and the timestamp. These three parts (data, timestamp, signature) are concatenated into a tuple. Any of the later processing steps (e.g., saving to persistent memory, transferring to the remote readout center) works on the tuple as a whole.

- *Wakeup Service (Wakeup)*. This pattern is an absolutely smart-meter-gateway-specific pattern. It implements the secure wakeup functionality, as required by the gateway protection profile (BSI, 2014). The remote entity sends a probe to the gateway. The gateway verifies the probe and responds with its own connection initiation to the remote entity if the authenticity of the probe is satisfied. If the probe is found to be unauthentic, the gateway simply drops the probe without any response.
- *Secure Communication (SC)*. In the common criteria protection profile of the smart meter gateway, a TLS protocol (Dierks and Rescorla, 2008) providing a secure communication channel between two communication partners is deemed mandatory for all connections between a gateway and a wide area network. TLS ensures that a handshake is performed between the gateway and the remote entity before any communication can occur. All the communication that occurs afterwards is encrypted using the algorithms and keys, etc., negotiated during the handshake. Thus, the data transmission occurs with authentic entities and is confidential. An important issue to note here is that DTLS (Datagram Transport Layer Security) (Rescorla and Modadugu, 2012) protocol can be used as alternative solution in the context of resource constrained systems.
- *Key Manager (KM)*. Within the gateway scenario, different keys have to be handled. The key manager pattern provides assistance to the application engineer to design a secure solution. The key manager uses different key rings, such as the public key ring and the private key ring. The keys from the key ring are used exclusively for their intended purposes.

6.1.5.3. System developer perspective: reuse of existing security patterns (Step 5 and Step 6) This process is relevant to both Step 5 (patterns identification and tailoring) and Step 6 (integration). The first activity in this process is to construct an access tool for the metrology domain, such as the one presented in Fig. 9.

In the security metrology process model, which is adapted from the IEC 61508 V-model (IEC, 2010), the developer begins with engineering requirements and subsequent system specifications. In each phase, the system developer executes the search/select task on the repository to tailor appropriate patterns to the modeling environment using the access tool and integrates these patterns into the application models by following an incremental process. Fig. 13 shows the simplified flow for an iteration within a software architecture definition phase. Moreover, the system developer can use the pattern designer tool (Arabion) to develop custom solutions when the repository fails to yield appropriate patterns during this phase.

The system design phase is the first phase impacted by our approach. In this phase, the high-level abstract system design is accomplished. The main task is to find solutions to the problem statement expressed by the requirements set. Use-case diagrams are defined to cover all the desired scenarios for which the future system has to provide functionality (see Fig. 11). For valuable insight regarding patterns that could possibly be used for the smart meter gateway, the application engineer uses the repository access tool to search for candidates. In this stage, no pattern has yet been instantiated; only information about the patterns is retrieved. An important step in the system design phase is the preparation of the security target document. This document is the basis for the common criteria evaluation in the certification phase.

The security concept diagram (see Fig. 14) provides a high-level architectural perspective in which major security requirements, techniques and concepts used to augment the security of the system are represented. The practical application of our approach begins at this point, where the system architect and security architect open the access tool and log in.

1. Both the system architect and the security architect analyze the system security requirements and identify the possible architectures and security techniques to be used. They begin defining the smart metering system. They identify the Smart Meter Gateway (SMGW) as a design pattern of interest to reach an EAL4+ via a security module technique (protection profile (BSI, 2014)). An interface separation (physically and logically), is explicitly required by the common criteria protection profile. To guarantee this mandatory separation, the smart meter gateway pattern provides the necessary system archi-

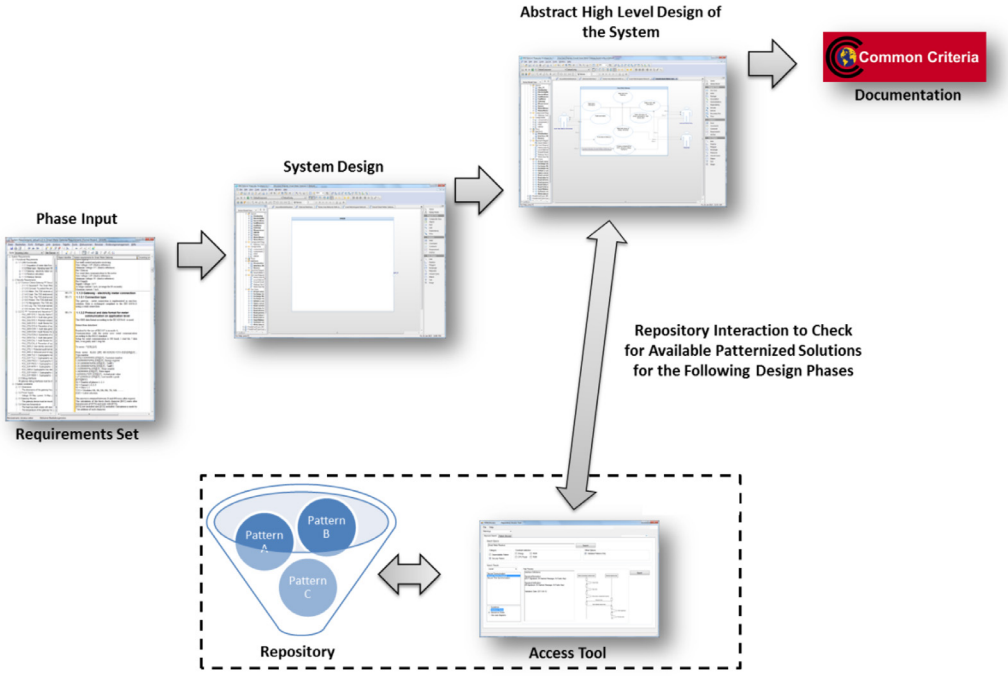

Fig. 13 – Simplified process flow using our approach in the software architecture definition phase.

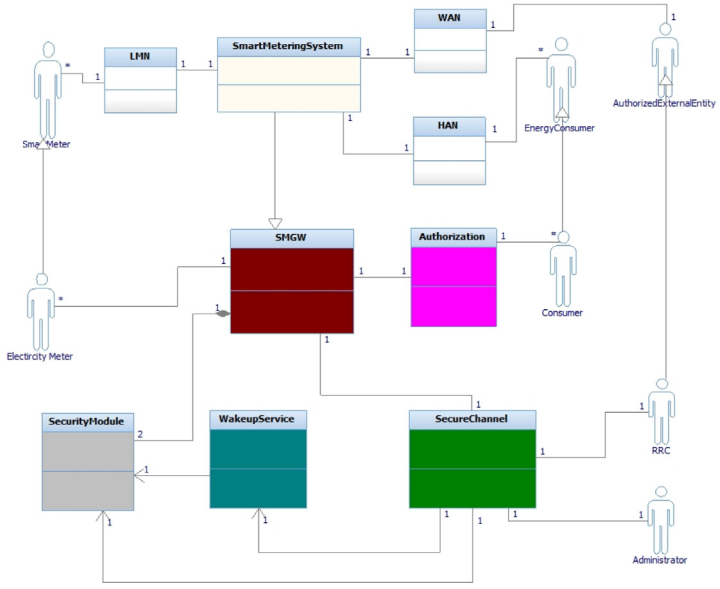

Fig. 14 – Overview of the smart meter security concept.

tecture skeleton, which is already included in high-level system design (see Fig. 12).

2. Both the system architect and the security architect continue refining the security concept (see Fig. 14), searching for design patterns in the access tools and importing them whenever a suitable design pattern is found:
   (a) The gateway requires an external security hardware module (IG_GW_SM) implemented with two independent security modules.
   (b) Initial decisions regarding the internal structure of the gateway (see Fig. 15):
      (a) A secure communication is selected to enable communication between the gateway and the AuthorizedExternalEntity. Therefore, a *secure channel pattern (also known as a secure communication pattern)* has to be integrated.
      (b) A wakeup service can be used to initiate a remote connection between the AuthorizedEx-

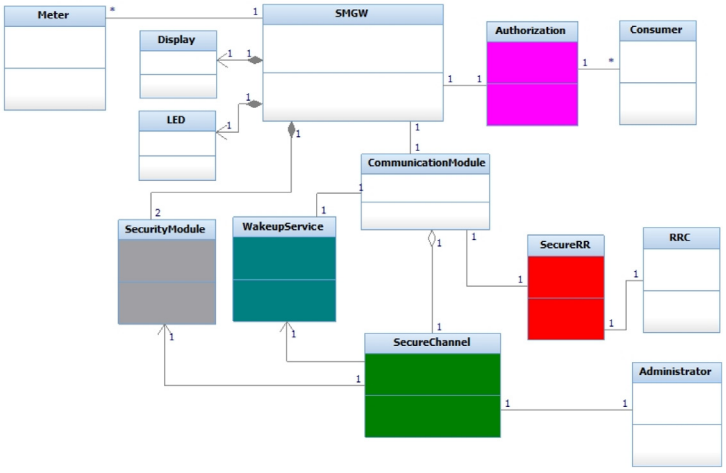

Fig. 15 – Overview of the smart meter system architecture.

ternalEntity and the gateway. Therefore, a *wakeup service pattern* has to be integrated.

(c) There are many roles requesting access to metering devices and the components of the metering system: It is necessary to execute access control for each function or action, which can be activated. For each function, it has to be specified which role is allowed to send a command. Therefore, an *authorization pattern* has to be integrated.

The system architecture shown in Fig. 15 specifies the smart meter gateway system decomposition and the relationship between the different blocks that compose the system. It follows the "separation of security-related systems from non-security-related systems" technique described in the protection profile. At this stage, the system architect makes several architectural decisions (based on the security concept and requirements) and accesses the repository to search for and import suitable refined and specialized design patterns of interest:

1. The SMGW is implemented with the following features:
   (a) The gateway's internal display (I_IF_DP): The data transmitted are the (1) overall amount of energy consumed; (2) actual consumption information (power); (3) version of the operating system; and (4) version of the gateway application.
   (b) The gateway LEDs (I_IF_LED): These provide information about the status, including (1) device online/offline; (2) secure channel established; and (3) wakeup mode.
   (c) The security module as a hardware component, such as that recommended in the protection profile.
   (d) A software application ("communicationModule") executed at the computing unit that integrates the secure channel as a pattern to support secure communication between the gateway and the remote read out center (RRC).
2. The wakeupservice as a pattern.
3. The secureChannel as a pattern.
4. The (security) remote readout center (IF_GW_RRC) as a pattern: This pattern provides information to the authorized external entities, including (1) values of the overall amount of energy consumed and (2) information about the actual power consumption. The remote readout center executes the remote readout security function and security techniques ("wakeupservice" and "secure channel"). The system remote readout function is based on three main functionalities (push a measurement of the actual consumption to the gateway, pull a measurement from the gateway, and wakeup service functionality).

After the specification of the abstract system design and the definition of use cases, the architecture design is accomplished as high-level class diagrams using the IBM Rational Rhapsody Developer. These diagrams are used to specify the static architecture of the gateway's software. In this phase, the application engineer queries the repository for suitable patterns using the access tool and the keyword-based search.

The software architect continues refining the software architecture. The secure remote readout function shown in Fig. 15 is refined, leading to the software architecture shown in Fig. 16. Then, additional software architectural decisions and analyses are made, and additional security techniques are identified. The software architect accesses the repository to search for and import suitable refined and specialized design patterns according to the identified techniques and integrates them. For instance, TLS is chosen as a domain-specific realization of the domain-independent secure channel. Hence, this choice results in the use of a random number generator pattern. Moreover, new patterns that are not represented in previous phases are initially introduced in this phase. This is the case of the key manager pattern, which is only of interest in this stage. When the software architect must define how to manage the required security keys, he/she can use the access tool to determine whether there is a pattern to implement this functionality. The key manager pattern is selected, imported and integrated into the software architecture diagrams.

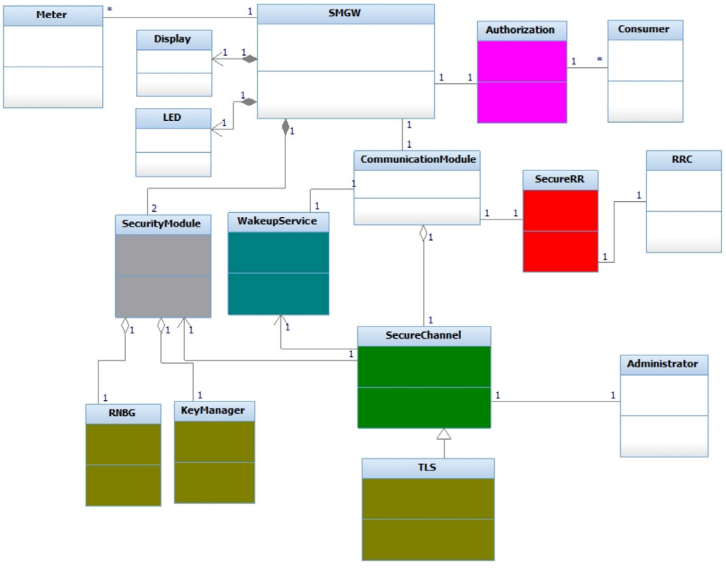

Fig. 16 – Overview of the smart meter software architecture.

6.1.6. *Discussion*

We discuss the results of the case study by answering the research questions introduced in Section 6.1.2.

6.1.6.1. RQ1 Is the approach feasible? In this paper, we presented some outcomes of our methodology for the creation of the PBSE modeling framework, as a set of methods and a set of modeling languages, for describing and modeling pattern-based system and software security development processes.

We show the feasibility of the approach through the construction of three artifacts: (1) the conceptual model - a set of concepts resulting from a security standard and best practices analysis; (2) the pattern modeling language - a domain-specific modeling language for defining patterns and for specifying the application of patterns for security design; (3) the PBSE method - a process for pattern-based development, including the selection and integration of patterns. These artifacts are complementary, and their integration represents a holistic approach to pattern-based development. We use MDE to describe these artifacts.

The development of these artifacts is not organized in chronological order, but we reported only the current versions. The methodologies for the creation of the modeling frameworks behind these artifacts follow the design science research method (Peffers et al., 2007). During design framework development, we define a set of main iteration cycles, including all key steps (which are described for each topic of concern) that are related to the three artifacts. Within each main iteration, micro-iterations between the different steps are employed to achieve our research goals. In the first main iteration, we deliver the first version of the conceptual model of security patterns. In the second main iteration, the modeling language is designed, and appropriate tools are developed to support these concepts. The results of these efforts are applied during the third main iteration to specify and define the set of patterns (e.g., in the form of models). The last main iteration is devoted to obtaining and tailoring suitable models to a form that is appropriate for the development/enactment environment. The results undergo a final complete evaluation at the end of each iteration, but each micro-iteration is also subject to evaluations. Based on the background of our research project partners, we start with an existing approach, such as the technology acceptance model (TAM) (Davis, 1989).

The conceptual model was presented at a workshop where the participants agreed that we had captured the most important concepts within security pattern practices. The DSMLs are discussed over several meetings with an expert in engineering software systems at the partner company where we were conducting the case study. After modification based on comments from the experts, these artifacts were used as inputs for the next steps.

The connecting piece between the security pattern repository and the development environment is the access tool. It can be customized for the existing development tools by deploying an appropriate back-end for pattern transformation. The back-end itself is composed of a transformation engine and transformation rules that transform pattern artifacts from the repository internal metamodel representation into that of the target environment model structure. The prerequisite for instantiating the patterns in the work space of the target tool is an import interface. Environments that provide import interfaces include, for instance, the Rational Rhapsody Developer from IBM and the Eclipse IDE with EMF support. Both offer a standard XML Metadata Interchange (XMI) interface for the import of modeling artifacts into the work space. The exchangeable back-end solution guarantees conformity of the pattern-based security engineering methodology with the existing tooling in companies. Companies do not need to change processes and tooling by equipping the access tool with an appropriate back-end for already existing development tool sets. This is very much appreciated since it has been stated as a requirement towards a possible solution. The methodology would not find wide acceptance in companies if

major parts of the existing engineering process and tool sets had to be aligned or changed. These results are confirmed by the survey presented in Section 6.2.

The proposed approach is non-intrusive with respect to the software engineering process because engineering roles that participate in the project (e.g., system architect) can develop the system with or without the use of our approach. If our approach is used, they can search for and import design patterns that have already developed for the tools already being used in the system and software engineering process. The engineering of a smart meter gateway demonstrates that all these concerns are feasible in a concrete industrial context. This finding is confirmed by the survey that we present in Section 6.2.

In summary, the feasibility in our study covers the ability to construct the conceptual model of security patterns, its use in creating DSMLs to specify patterns and the development of tools to support the development of pattern-based secure systems. Furthermore, we show how the proposed methodology can be adapted to the existing engineering process and integrated into the existing tool chains. Through the smart meter gateway case study we have shown that the construction of all these artifacts are feasible in a concrete industrial context.

6.1.6.2. RQ2 Is the effort involved in the application of the approach acceptable? The creation of the conceptual model required approximately 4 person months. The majority of the effort was spent on eliciting, defining and then modeling the concepts and their relationships based on the standards and state-of-the-art pattern modeling and formalization. The rest of the time was mainly used to review and revise the proposed conceptual model. The creation of the modeling language (abstract and concrete syntaxes) for the security pattern required approximately 6 person months. This included the effort made to understand the usage of EMFT to specify metamodels in Ecore and to generate other representations.

The implementation of the MDE tool chain required 12 person months. The most important part of the effort was spent on understanding the CDO repository platform, including the architecture, functionalities and deployment mechanisms. The remaining time was primarily used to develop a customized access tool for the metrology development process, including the development of a set of model transformations. The proposed tool chain was designed to support the proposed metamodels; hence, the tool chain and the remainder of the activities involved in the approach could be developed in parallel. This activity needed to be performed only once for the development of a given set of applications. In addition, we expect the effort for the creation of a DSML and the development of tools to be less for future applications, as we had to address several technical details in relation to using EMFT and CDO in our first application.

The process of populating the repository using the MDE tool chain took approximatively three months. It should be noted, though, that most of these steps may be re-used to develop other systems.

The next task was the construction of the domain model (i.e., smart meter gateway system model), which was completed within 3 person months. Finally, the process of identifying, tailoring and integrating the patterns was completed within three months. This process required comparatively less effort. This is partly due to the features provided by the MDE tool chain and to the support provided by the experts at the partner company.

There are several degrees of reuse in our work. Some components of the approach itself may be reused, such as explained during the high-level description of the approach at the beginning of Section 4. For example, Step 3 is performed once for a set of domains, while Step 5 is performed once for each development environment. We believe that the specification and packaging of security patterns will foster technology reuse across domains (the reuse of models at different levels). Such reuse would involve, in the first instance, customizing and reinstantiating the patterns in new contexts (which, in any case, would have to be similar, e.g., the patterns for an electricity meter gateway would apply to other smart meter applications with additional features), as well as introducing new patterns as befits the situation. In addition, our model-based specification and packaging of patterns and its related tooling are more useful in identifying suitable security patterns to be used in the software architecture of an application than the security solutions documentation itself. This was also reflected during the survey (see Section 6.2). The goal is to improve software design automation using patterns instead of just imitating them during the design.

The criterion used by the experts for answering **RQ2** was that the (perceived) cost savings arising from the using of the approach should reasonably exceed the cost of applying the approach. Therefore, the estimations given for the engineering of secure systems using patterns took into account several factors. These included the current time span for the development of a typical security component (between 2 and 6 months), the effort required to be invested into the preparation of secure system project description documents and the management processes, the costs associated with the involvement of an external security expert, and the side benefits that the models built in our approach could helps to shorten down certification process and training periods of newly employed engineers.

The purpose of this research question was to assess whether the cost savings arising from the use of the approach exceed the cost of applying the approach. Therefore, the estimations given for the metrology domain took into account two different scenarios with associated argumentations.

Scenario 1: International metrology manufacturing company In the first scenario, an international smart meter and smart meter component manufacturing company is the main focus. It serves the metrology market worldwide, including its main markets: the USA and Europe. The company produces meters for different commodities (electricity, gas, water, heat, etc.) in different sizes (from private households to large-scale consumers) for the mentioned markets. It has specialized divisions for the different meters. The different legislative backgrounds of the company's main markets play an important role in the development of new devices to serve the needs of the markets. Different development teams work on meters for these markets. The company has a team of security experts. They are part of the development team for most of the products since security plays a more important role in this domain. However, security experts are expensive, and their

role in the development process is sometimes only to offer minor assistance regarding a specific security question. Nevertheless, they are part of the team and have to take part in all the meetings. This could be optimized through efficient use of their time, having them concentrate only on security problems and provide their expertise without taking part in every meeting. Another reason is that the market of experienced security experts is small and that the company is not always able to employ enough experts for their different product lines. Therefore, the reuse of components (hardware and software) throughout the development processes of the devices is of main importance. To serve this need, the company owns a proprietary kind of repository for sharing documents and knowledge between the different divisions. It is used to reduce the development time and costs and to avoid reinventing existing solutions for every new product.

The experts involved in the use case felt that the effort spent on creating the conceptual model of security patterns and on the development of the DSMLs was slightly high. However, these two tasks were performed one time for a set of domains. Therefore, they found the level of effort to be acceptable. The argumentation here was that the resulting modeling framework (1) reduces the time/cost of product development, with the reuse of design patterns to address security issues; (2) reduces the time to market for new metering devices since evaluation and certification are time-consuming processes, which may be sped up by having detailed and clear product development documentation. This may be directly attributed to the type approval of a device. The type approval procedure may be sped up by providing detailed product documentation, especially for the security-related parts; and (3) reduces the number of full time security experts by providing security knowledge with patterns since experienced security experts in this domain are rare and cost intensive. Thus, this effort can be spread over a number of projects, as the same set of security patterns are used for a large number of smart meter production systems.

Scenario 2: SME company In contrast to the international meter manufacturing company of scenario 1, the second scenario presents an SME meter manufacturer specializing in electricity meters. The company mainly serves a single country market. It has experienced engineers for voltage and current sensors and embedded systems but no security experts. The company is highly interested in developing a smart meter gateway and to be one of the first providers in the market. However, engineers are expensive. The company is in direct competition with other meter manufacturers and needs to optimize and reduce time and cost for product development. The development team is able to work on the functional part of a gateway device, but it needs profound security knowledge to successfully implement the requested security functionality. Another problem is the availability of experienced engineers. It may occur that one of the engineers leaves the company during an unfinished development cycle of a product. In this case, his gained knowledge is obsolete with respect to the company and cannot be handed over to his successor to reduce the training period. To address this kind of situation, a kind of repository that can store structured knowledge is of great benefit.

As in the previous scenario, the experts involved in the use case generally found the level of effort spent creating the modeling framework to be acceptable. The argumentation here was that the resulting modeling framework (1) reduces product development cost and time, with the reuse of design patterns to address security issues; (2) offers a competitive advantage to other manufacturers and market participants in terms of having expert security solutions in their products that have been evaluated in depth; (3) reduces the time to market for new metering devices since evaluation and certification are time-consuming processes, which may be sped up by having detailed and clear product development documentation. This may be directly attributed to the type approval of a device. The type approval procedure may be sped up by providing detailed product documentation, especially for the security-related parts; and (4) keeps structured expert knowledge inside the company, even if employees change, by using a repository with patterns. This also helps to shorten the training periods of newly employed engineers.

6.2. *Survey*

After the completion of our case study, we conducted an experiment in which we presented our approach and the solution of our case study to collect feedback from industry practitioners through a survey. We employed the factors developed in Rogers' theory of innovation diffusion (Rogers, 2003), which were involved in technology adoption: (1) Trialability; (2) compatibility; (3) relative advantage; (4) observability; and (5) complexity. Twenty people attended the TERESA MDE workshop in Toulouse, where the experiment was initiated. All participants were recognized as experts in their domains, with a high-level of skill in engineering secure systems, who had already participated in the development of several projects related to their skills.

6.2.1. *Context and data generation method*

To carry out the survey, an MDE workshop is organized to provide a reasonably thorough overview of the proposed approach to the experts. The participants were trained to use our method and the tool suite. The training was conducted in two sessions on the same day. The first session was 3 hours long and was managed by two instructors. The first instructor introduced MDE and presented a pattern-based development methodology and how it might be used to support the development of secure applications. In addition, a 1-hour practice session on Eclipse and the EMFT environment was presented by the second instructor as a laboratory exercise. During the second hour-long session, several operating examples, with detailed explanations, were introduced to the participants by two additional instructors who participated in the development of the tool suite. Finally, a set of materials was provided to the participants: the tool suite and its accompanying installation and user documentation, a detailed textual description of the patterns and their properties, a detailed requirements document, a set of models describing the system under development and the used patterns in the form of UML diagrams. Finally, the participants were given a subjective post-experiment questionnaire consisting of a set of questions, as described in the following section, and space for comments.

The questionnaire was anonymous and divided into four parts. The questionnaire was uploaded online using Google Docs, and the link to the questionnaire was forwarded to the participants. All of the registered participants received the questionnaire link in this manner. The context of the experiment, including the description of the methodology for the experiment, participant selection, data collection and statistical analyses, was clearly defined to exclude bias due to relations to the authors. The participants also knew that the experimenter did not have any influence on their job evaluation scores or their career evolution and progress, as the experimenter was external with respect to their organizations. Statistical analyses, mainly for the averages of the values provided by the participants, were performed by peers (independent of the experimenter) on anonymized data (both subjects and scales). Moreover, the researcher agreed to use the survey only for research purposes as anonymized results.

6.2.2. *Questionnaire*

We developed a simple questionnaire to address and measure the interest of practitioners regarding engineering secure systems using patterns and models. The questionnaire was divided into four parts. The following items present an excerpt of the questionnaire presented to the participants, who were asked to rate their satisfaction on a Likert-like scale (Likert, 1932). In the context of this experiment, we have defined a scale from 1 to 5, 1 being the lowest value of satisfaction or the greatest difficulty (i.e., *Very Difficult, Very Likely Not*) and 5 being the highest value of satisfaction regarding the presented concepts or the greatest ease in realizing a solution to the given question (i.e., *Very Easy, Definitely*).

The first section, which consisted of Q1-Q2, as shown below, was concerned with the backgrounds of the subjects related to software engineering practices in general and to security in particular to show on what kind of experience these experts based their answers.

- *Q1: Is security in software engineering an important aspect of your job?*
  - ☐ *Yes*
  - ☐ *No*
- *Q2: How much experience do you have with security in software-engineering-related activities?*
  - ☐ *Less than 6 Months*
  - ☐ *More than 6 months but less than 12 months*
  - ☐ *More than 1 year but less than 2 years*
  - ☐ *More than 2 years*

The second part, Q3-Q7, was concerned with the subjects' experience with the engineering of secure systems. Although we based our work on prior knowledge regarding the difficulty in using existing methods, practices and standards, we wanted to ensure that this was also the case in this group.

- ☐ *Q3: Is your own work (current or past) related to the development of secure systems?*
  - ☐ *Yes*
  - ☐ *No*
- *Q4: Have you participated in the development/purchase process of a secure system?*
  - ☐ *Development*
  - ☐ *Purchase*
  - ☐ *Both*
  - ☐ *Neither*
- *Q5: Based on your experience, how easy it is to build a secure system?*
- *Q6: Do you have skills in some non-model-driven engineering-related approaches to support the development of secure systems?*
  - ☐ *Yes*
  - ☐ *No*
- *Q7: Based on your experience, is this approach easy to use for developing secure systems?*

The third part, Q8-Q11, was concerned with the modeling of secure systems. In particular, we wanted to assess the perception of using the modeling approaches to develop a secure system.

- *Q8: Was the presented conceptual model easy to understand?*
- *Q9: Do you find the models simple enough to use for communication within an overall engineering secure system project?*
- *Q10: How easy was it to integrate the resulting concepts (tool suite) into your favorite development environment?*
- *Q11: Would you like to use the entire approach or some of its steps in the future?*

The final part, Q12-Q14, was concerned with the feedback regarding the overall approach and the use of model-driven engineering.

- *Q12: Overall, how easy was it to follow the steps of our approach?*
- *Q13: Would you see value in adopting the presented approach at your company for the development/purchase of a secure system?*
- *Q14: Does the presented tool suite provide useful assistance in the development of secure systems?*

6.2.3. *Survey results*

Out of the 20 invitees, 16 completed the questionnaire, yielding a response rate of approximately 80%. 75% had practical experience in security critical system development. Sixty-nine percent had practical experience in the application of MDE to engineering software in general, and 62%, to engineering secure systems. Moreover, six software engineering experts participated in the survey. Six security experts participated in the survey. In addition, four project managers participated in the survey. A broad range of industry sectors was represented, with respondents from the metrology, railway, automotive and software development sectors. The majority of respondents belong to organizations with 50 people or more.

Based on the responses obtained, security in software engineering was an important aspect of the job for all but two participants (Table 3, *row a*). Overall, 56% of the participants had over two years of experience with security in software engineering, while 13% had at least one year of experience with security in software engineering (Fig. 17).

Regarding the experience of the participants with the development/purchase of secure systems, all except three had

Table 3 – Answers to questions Q1, Q3 and Q6.

	Questions	Answers	
		Yes (%)	*No (%)*
a	*(Q1) Is security in software engineering an important aspect of your job?*	88	13
b	*(Q3) Is your own work (current or past) related to the development of secure systems?*	81	19
c	*(Q6) Do you have skills in some non-model-driven engineering-related approaches to support the development of secure systems?*	69	31

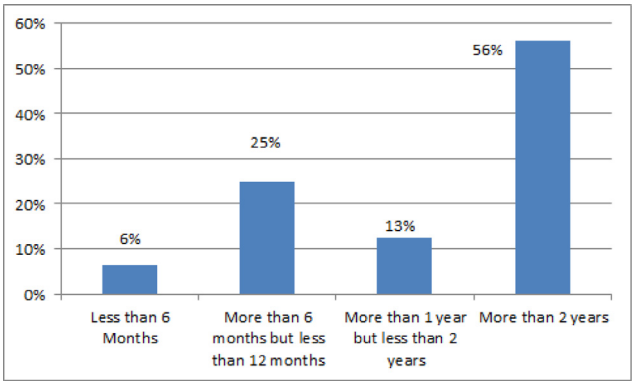

Fig. 17 – (Q2) How much experience do you have with security in software engineering related activities?.

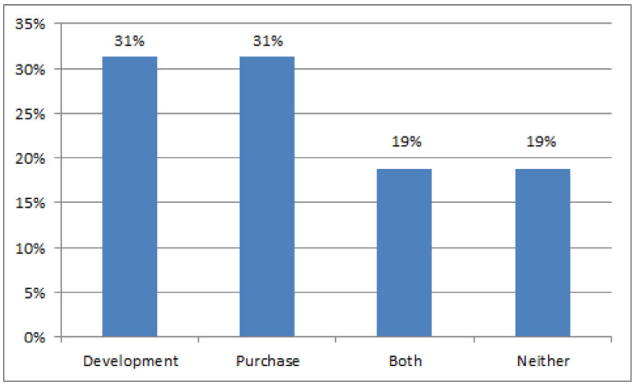

Fig. 18 – (Q4) Have you participated in the development/purchase process of a secure system?.

worked in this context (Table 3, *row b*). From the set of participants who had participated in the development/purchase of a secure system, 31% had participated in the development of a secure system, 31% had participated in the purchase of a secure system, and 19% had participated in both activities (Fig. 18).

Table 4, *row a* shows that 63% of the participants felt that building a secure system was difficult, while the rest thought that it was of average difficulty. Regarding the skills related to engineering secure system approaches, 69% of the participants had worked with non-model-driven engineering-related approaches to support the development of secure systems (Table 3, *row c*).

Table 4, *row b* reveals that 31% of the participants perceived this approach to be very easy to follow, whereas 50% thought it was easy, and the remaining 19% experienced average difficulty. When presented with the conceptual models of the approach, 31% of the participants perceived the approach to be being very easy to understand, whereas 44% thought it was easy to understand, and the remaining 13% experienced average difficulty (Table 4, *row c*).

All participants agreed that the models created during the application of a pattern-based and a model-driven engineering approach were simple enough to use in communication within an overall engineering secure system project (Table 5, *row a*). Moreover, the participants thought that it would be beneficial to use within their industries (Table 4, *row d*).

Table 5, *row b* shows the extent to which the participants indicated that they would likely use the pattern-based approach in the future for the development of other kinds of secure systems. Of the participants in the experiment, 25% of them indicated that they would definitely continue their application of the approach, whereas 56% said that they would very probably do so.

Table 4, *row e* reveals that 31% of the participants perceived the approach to be very easy to follow, whereas 50% thought that it was easy, and the remaining 19% experienced average difficulty.

With regard to the adoption of the approach, 25% of the participants thought that there definitely was value in adopting the approach, and a further 56% thought that the approach was very likely worth adopting (Table 5, *row c*). The current tool suite based on EMFT was also thought to be useful: 31% of the participants believed that the tool would definitely be useful for engineering a secure system and a further 50% thought that it would very likely be useful. The remaining 19% of the participants thought that it would probably be useful (Table 5, *row d*).

6.2.4. *Discussion*

In summary, the answers received in our survey suggest that the proposed approach was overall regarded as easy to learn and follow. These responses indicate that a pattern- and model-based development method for addressing secure systems through a model-driven engineering approach should be investigated further. Because the pattern conceptual model can be applied to multiple problems through a simple extension, these results suggest that our work has wider applicability and usefulness.

7. Fulfillment of the security objectives

This section proves that the set of identified security patterns is suited to fulfill the desired security objectives defined in form of requirements to protect the assets. This suggested the following question, raised as a third research question:

- RQ3. *Can we show that a system built based on the use of security patterns is secure?*.

Table 4 – Answers to questions Q5, Q7, Q8, Q10 and Q12.

	Questions	Answers				
		Very Difficult (%)	*Difficult (%)*	*Average (%)*	*Easy (%)*	*Very Easy (%)*
a	*(Q5) Based on your experience, how easy it is to build a secure system?*	63	25	13	0	0%
b	*(Q7) Based on your experience, is this approach easy to use for developing secure systems?*	0	0	19	50	31
c	*(Q8) Was the presented conceptual model easy to understand?*	0	13	13	44	31
d	*(Q10) How easy was it to integrate the resulting concepts (tool suite) into your favorite development environment?*	0	6	19	50	25
e	*(Q12) Overall, how easy was it to follow the steps of our approach?*	0	0	19	50	31

Table 5 – Answers to questions Q9, Q11, Q13 and Q14.

	Questions	Answers				
		Very probably (%)	*Probably not (%)*	*Probably (%)*	*Very probably (%)*	*Definitely (%)*
a	*(Q9) Do you find the models (patterns) simple enough to use for communication within an overall engineering secure system project?*	0	0	19	63	19
b	*(Q11) Would you like to use the entire approach or some of its steps in the future?*	0	0	19	56	25
c	*(Q13) Would you see value in adopting the presented approach at your company for the development/purchase of a secure system?*	0	0	19	56	25
d	*(Q14) Does the presented tool suite provide useful assistance in the development of secure systems?*	0	0	19	50	31

In the context of our experiment, the security requirements are specified in terms of a set of desirable security properties (i.e., positive statements) using common taxonomies such as CIA. Patterns are then introduced according to expected security properties. For instance, the Common Criteria Protection Profile (BSI, 2014) recommends a set of measures and techniques that can be employed as patterns to stop or mitigate the identified threats for the whole smart metering system and the threats against the smart meter gateway, in particular. In this way we can build, for example, secure gateway systems in an integrated way, including measurement data, configuration data, and other related information, not just isolated networks; the same is true for safety-critical systems or other types of applications.

As described above, the smart meter gateway serves as connection piece between different networks. Seen as first line of defense, communication channel between smart meter gateway and an authorized external entity, e.g., the remote readout center (RRC), is a main point of attack. The information transmitted and received via the above mentioned channel or the channel itself may be manipulated by an attacker. Indeed, communication channels are main attack point, but physical attacks on the gateway and its security module including the key material are also possible. Note that these threats include external and internal attacks. In Table 6, we used STRIDE taxonomy to describe some of the possible scenarios of software and network attacks. More detailed information on the gateways security threats, attack paths and security objectives can be found in the smart meter gateway protection profile (BSI, 2014).

To limit the threats listed in Table 6, we have two types of security mechanisms. For the CIA the use of encryption algorithms and message authentication code is mandatory; this can be partially found in the SSL/TLS specification for application level security, such as that recommended in the protection profile. For protection against unauthorized access, we will rely mainly on access control techniques (e.g., authorization pattern).

The case study in Section 6.1 was conducted to demonstrate that a non-trivial system, that was initially developed independently of our approach can retroactively be modeled using our pattern-based methodology. During the application of the approach, presented in Section 4, a number of threats in the initial models have been found and mitigated using already defined patterns. We provide analyzes approach of stopping the attack by enumerating possible security patterns that can be applied for this purpose. Our added value is that the analysis does not only check if the needed security mechanisms exist but also that they are correctly used in form of security patterns to stop or mitigate the threats. This is an

Table 6 – Incomplete attack scenarios against communication channel between the Gateway and an authorized external entity.

Attack scenario	Threat type	Security property
An attacker pretends to be the RRC.	Spoofing	Authenticity
An attacker changes the measurement data or configuration data when transmitted between the Gateway and an external entity in the WAN.	Tampering	Integrity
An attacker gains access to the measurement data or configuration data or parts of it when transmitted between Gateway and external entities in the WAN.	Information disclosure	Confidentiality
An attacker gains customer access to the Gateway, due to missing authorization.	Elevation of privileges	Authorization

important benefit, as the proposed approach can help to improve the quality of a system's current documentation. It also demonstrated that each of the patterns were variable enough to deal with multiple contexts, where the respective concrete patterns were applicable (e.g., secure communication pattern). Note that compared to other safety-critical areas, such as defense or nuclear power generation, no injuries are caused if the smart meter gateway is offline. However, our vision is not limited to engineering secure systems. We have also designed secure architectures for SIL4 safety-critical embedded systems for railway signaling (ERTMS/ETCS) (Hamid and Perez, 2016).

8. Recapitulation and synthesis

From the system developer perspective, security issues not only are detected in code and need to be identified early in the first development steps and at the highest levels, primarily in the architecture design stage, where their semantics are clear, they are also mapped to lower levels, where they are enforced by the corresponding concrete mechanisms. For instance, the MDSN SDLC (Microsoft, 2012) addresses some of these issues and provides support for using threat models and abuse case modeling during the design stage of the SDLC. To build secure systems, Neumann (2004) expressed the need to develop principled systems that are based on solid conceptual approaches. Recently, Fernandez (2013) adapted Neumann's discovery to stress that building secure systems requires the careful application of well-defined principles. Patterns enable the implicit application of principles, and at the same time, patterns provide a systematic approach to describing best practices. The design framework that we built helps in this respect. We employ the MDE and DSML technologies and attempt to add more formality to improve parts of the system design.

Security solutions can be described as security patterns, and the use of these patterns results in products that are already established in that domain, usually in the form of COTS components. Patterns define best practices, and the goal is to help designers reuse them in new designs. Security requirements are specified independent from patterns and technological products at a very early stage of system design and then refined using the system model until they can be matched with security patterns. The main problem for a developer is to select and connect security patterns with the rest of an application. Making some aspects of the pattern description precise would make these aspects more convenient. Even if a pattern requires tailoring, starting from a precise description facilitates its selection and application. A security pattern targets some particular properties that characterize it, and the integration, composition and application of a security pattern should maintain these properties.

In our work, we propose a pattern- and model-based development method for addressing security through a model-driven engineering approach. The approach is composed of several steps and is based on metamodeling techniques that enable the specification of patterns. It is also based on model transformation techniques for the purposes of generation. The defined conceptual model points to a common representation for several contexts of use. First, this approach aims to allow design automation through the reuse of security applications captured in the form of patterns. Second, it aims to overcome the lack of formalism in a conventional text-based approach, supporting model-based analysis and verification techniques. The approach empowers system and software engineers to reuse solutions for the engineering of secure systems without specific knowledge of how the solution is designed and implemented. The resulting modeling framework reduces the time/cost of understanding and analyzing system artifact descriptions by virtue of its abstraction mechanisms and reduces the cost of the development process by virtue of its generation mechanisms.

The pattern formalization approach is very ambitious. In fact, a rigorous treatment of security properties needs to be based on clear formal semantics that enable system developers to precisely specify security requirements. Thus, the precise but flexible specification and description of security patterns are pre-requisites to their successful integration and composition for their application. The ultimate goal of our approach is to ensure that the defined patterns are applied in a way that has previously been demonstrated to be correct and useful in secure design. Analysis of patterns can be performed on a more abstract level, where their security properties' semantics are clear, and then more refined security properties corresponding to a more concrete level can be derived through transformation techniques. In our work, we design a modeling language supporting two complementary types of representations: a semi-formal representation through metamodeling techniques and a rigorous formal representation through formal language. It provides a generic setup to formalize a substantial number of security patterns. It introduces much (formal) machinery, where the actual usefulness is demonstrated through a set of experimental evaluation scenarios. The application of a PBSE methodology requires a relatively complete

catalog of patterns, covering all architectural levels, as well as computation, communications, and control aspects. An important number of security patterns have been produced, but we need to tailor them and to provide more. This will raise the abstraction levels to design and reason regarding security solutions for the developers of security applications and/or security application practices and thus make it easier to apply security solutions correctly, even with limited technical know-how. End users (e.g., security management operators) will benefit from MDE automation of the patterns and consequent offloading of the security tasks to the application and platform infrastructure.

9. Related work

Over the years, research efforts have been invested in methodologies and techniques for secure software engineering, although dedicated processes have been proposed only recently (B. De Win et al., 2009; Barnabe et al., 2011). A survey is presented in Uzunov et al. (2013). In our work, our aim is capturing and providing this expertise by the way of adding security patterns directly into application models, targeting the (i) development of an extendible design language for modeling patterns in secure distributed embedded systems (Hamid et al., 2011) and (ii) a methodology to improve existing development processes using patterns (Hamid et al., 2013). The language must capture the core elements of the pattern to support its (a) precise specification, (b) appropriate selection and (c) seamless integration and use. The first aspect is related to pattern definition, whereas the second and third aspects are more related to problem definition. From the pattern-based system and software engineering methodological perspective, only a few works (Abowd et al., 1995; Soundarajan and Hallstrom, 2004; Zdun and Avgeriou, 2008) have addressed this concern. They are harmonized with the use of patterns in each system and software development lifecycle stage. However, existing approaches using patterns often target one stage of development (architecture, design or implementation) due to the lack of formalisms ensuring (1) the specification of a pattern at different levels of abstraction, (2) relationships that govern their interactions and complementarity and (3) the relationship between patterns and other artifacts manipulated during the development lifecycle and those related to the assessment of critical systems.

Several approaches exist in the security design pattern literature (Cheng et al., 2003; Fernandez et al., 2011; Giacomo et al., 2008; Hatebur et al., 2007; Jürjens et al., 2002; Katt et al., 2013; Schumacher, 2003). They allow solutions to 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 and authorization. In Hatebur et al. (2007), authors have defined a pattern-based security requirements engineering method that is applicable after the security goals and an initial set of security requirements are elicited. The approach does not consider the elicitation of the initial set of security requirements, since only dependent security requirements are elicited. Works from authors in Schumacher (2003) present a detailed description of patterns used in security engineering in different domain. These include patterns on risk management such as patterns on Enterprise Security Approaches and Threat Assessment. These patterns are to be integrated in methodologies and will, for example, guide an enterprise in selecting security approaches or lead the engineer to use patterns during the development life cycle to resolve recurring security challenges. In a previous work (Hamid et al., 2013), we considered building a security-oriented Pattern-Based System and Software Engineering life cycle based on a repository of security patterns. The central idea of the approach is to assist the system designer through interactions with a repository of security patterns in resolving recurrent security problems at the right moment in a model-based development life cycle.

Existing formalization attempts for patterns (Mikkonen, 1998; Soundarajan and Hallstrom, 2004) fall short in handling the inherent variability in pattern descriptions (Zdun and Avgeriou, 2008), and they focus primarily on a very limited design and architecture pattern scope. They do not yet address specific domains, such as security and safety. For the first type of approach (Gamma et al., 1995), design patterns are usually represented by diagrams with specific notations, such as UML object diagrams, that are 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 fact, the existing descriptions are not formal definitions and sometimes leave ambiguities regarding the exact meaning of the patterns. There are some promising and well-proven approaches (Douglass, 1998) based on Gamma et al. (1995). However, this type of technique does not afford the high degree of flexibility in the pattern structure that is required to reach our objectives. Thus far, patterns have been used in systematic engineering approaches for various tasks, such as classification and organization, pattern selection based on security requirements (Hafiz et al., 2007; Weiss and Mouratidis, 2008), analyzing and modeling security requirements (Cheng et al., 2003), and measuring the introduced security level (Fernandez et al., 2010).

The recently completed FP6 SERENITY project has introduced a new notion of security and dependability (S&D) patterns. SERENITY's S&D patterns are precise specifications of validated S&D mechanisms including a precise behavioral description, references to the S&D properties, constraints on the context required for deployment, information describing how to adapt and monitor the mechanism, and trust mechanisms. The S&D SERENITY pattern is specified following several levels of abstraction to bridge the gap between abstract solution and implementation. These abstraction levels are S&D classes, S&D patterns and S&D implementation. Such validated S&D patterns and the formal characterization of their behavior and semantics can also be the basic building blocks for S&D engineering in embedded systems. Serrano et al. (2008) explained how this can be achieved using a library of precisely described and formally verified security and dependability (S&D) solutions, i.e., S&D classes, S&D patterns, S&D implementation and S&D integration schemes. Moreover, Giacomo et al. (2008) reported an empirical experience regarding the adoption and elicitation of S&D patterns in the air traffic management (ATM) domain, demonstrating the power of using patterns as guidance to structure the analysis of operational

aspects when used at the design stage. Conceptually, our modeling framework is similar to that proposed in the SERENITY project. Nevertheless, the pattern structure is rigid (a pattern is defined as quadruplet) and is thus unusable for capturing specific characteristics of S&D patterns. However, the SERENITY project proposes several levels of abstraction to bridge the gap between abstract solution and implementation, which intends to not capture a common representation of patterns for several domains.

Usually, these design artifacts are provided as a library of models (sub-systems) and as a system of patterns (framework) in the more elaborate approaches. However, there remains a lack of modeling languages and/or formalisms dedicated to specifying these design artifacts and understanding their reuse in software development automation. More precisely, a gap between the development of systems using patterns and the pattern information remains. Most patterns are expressed in a textual form, as informal indications on how to solve individual design problems. Some of them use more precise representations based on UML diagrams, although these patterns do not include sufficient semantic descriptions to automate their processing and to extend their use. Furthermore, the correct application of a pattern is not guaranteed because the description does not consider the effects of interactions, adaptation and combination, making them inappropriate for automated processing within a tool-supported development process. Finally, due to manual pattern implementation, the problem of incorrect implementation (the most important source of safety issues) remains unresolved.

10. Conclusion

The proposed approach for engineering secure systems is dependent on patterns and models as first-class citizens to specify applications within a particular domain and focuses on the problem of software system engineering using a design philosophy that fosters reuse. This approach may significantly reduce the cost of engineering a system because it enables security issues to be addressed early in the system development process while simultaneously relieving the developer of the technical details. We begin by specifying a conceptual model of the desired patterns and proceed by designing modeling languages that are appropriate for the content. The results of these efforts are subsequently employed to specify and define a security solution as a pattern (e.g., in the form of design diagrams). Developing an application using pattern-based development processes and reusing existing patterns requires finding and tailoring suitable patterns to a form that is appropriate for the targeted development environment. The integration phase of our approach enables a domain engineer to reuse the resultant patterns that have been previously adapted and transformed for a given engineering environment (development platform) to develop a domain-specific application.

In addition, we provide an operational architecture for a tool suite to support the proposed approach. An example of this tool suite, which is termed Semcomdt, is constructed using EMFT and a CDO-based repository and is currently provided in the form of Eclipse plugins. In addition, the tool suite promotes the separation of concerns during the development process by distinguishing the stakeholder roles. Access to the repository is customized with regard to the development phases, the stakeholders' domain and system knowledge. We evaluate the usefulness of the patterns for increasing engineering productivity. We intend to demonstrate that the security pattern-based approach generates a reduced number or a simplification of the engineering process steps. The design solutions that are provided should support developers regarding security issues and reduce the error frequency. We demonstrate that the application of the proposed approach yields important benefits to development engineers. This fact is demonstrated via the implementation of a demonstrator. Initial evidence from a survey reveals that domain experts perceived the approach to be extremely useful and agreed regarding the benefits of adopting the approach in a real industrial context.

In our future work, we plan to study the automation of the model search and tailoring tasks. Our vision is for patterns to be inferred from the browsing history of users and constructed from a set of previously developed applications. As we look to the future, we can employ existing studies on reuse scenarios and design space exploration (Hamid, 2015; Hegedüs et al., 2015; Tomer et al., 2004). We would also like to study the integration of our tools with other MDE tools. The objective is to show the process flow and the integration of the tools in the domain tool chains, whereas the intention is not to resolve the low-level details of the approach integration. We must implement other types of software and means of generating validated artifacts, such as programming language code and certification artifacts, which are capable of producing a restrictive set of artifacts that comply with domain standards.

Concurrently, more sophisticated techniques for deriving artifact relationships can be implemented using different domains to reduce the complexity of designing systems of patterns. We will seek new opportunities to apply the proposed approach to other domains. This task requires an instantiation of the complete software engineering tool and method and an evaluation of the experiences of many users across many domains. We would like to enhance the proposed integration process by automating the detection of conflicts between the pattern structure and the existing application architecture and propose solutions in a manner similar to the way in which merging tools operate.

Another short-term objective is to teach the engineering of secure systems and the effect of using patterns and models to design these systems, as well as the effect of security and reuse practices on the total system and software quality, which is related to further assessment and usability testing in industry, industry-like and research projects.

Acknowledgments

This work was initiated within the context of the SEMCO project. It was supported by the European FP7 TERESA project and by the French FUI 7 SIRSEC project. Particular thanks go to Adel Ziani and Jacob Geisel for their valuable assistance in the implementation and development of the SEMCO tools. In addition, we would like to thank the TERESA consortium mem-

bers for their participation in the implementation of the case study and the survey.

REFERENCES

Abowd G, Allen R, Garlan D. Formalizing style to understand descriptions of software architecture. ACM Trans Softw Eng Methodol 1995;4(4):319–64.

Alexander C, Ishikawa S, Silverstein M. Center for Environmental Structure Series, 2. Oxford University Press; 1977. ISBN 9780195019193

Alvi AK, Zulkernine M. A natural classification scheme for software security patterns. Proceedings of IEEE ninth international conference on dependable, autonomic and secure computing (DASC). IEEE; 2011. p. 113–20.

Anderson R. Security engineering: a guide to building dependable distributed systems. 2nd. Wiley; 2008.

Anwar Z, Yurcik W, Johnson RE, Hafiz M, Campbell RH. Multiple design patterns for voice over IP (VOIP) security. Proceedings of 2006 IEEE international performance computing and communications conference; 2006. p. 485–92.

Atkinson C, Kühne T. Model-driven development: a metamodeling foundation. IEEE Softw 2003;20(5):36–41.

Avizienis A, Laprie JC, Randell B, Landwehr C. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Depend Secur Comput 2004;1:11–33.

Barker E, Kelsey J, 2012. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. National Institute of Standards and Technology.

Barnabe D, Goodnight J, Hamilton D, Bayuk JL. Systems security engineering. IEEE Secur Priv Mag 2011;9:72–4.

Basin D, Clavel M, Doser J, Egea M. Automated analysis of security-design models. Inf Softw Technol 2009;51:815–31.

Basin D, Doser J, Lodderstedt T. Model driven security: From UML models to access control infrastructures. ACM Trans Softw Eng Methodol (TOSEM) 2006;5(1):39–91.

Braber F, Hogganvik I, Lund M, Stølen K, Vraalsen F. Model-based security analysis in seven steps - a guided tour to the CORAS method. BT Technol J 2007;25(1):101–17.

BSI, 2014. Protection Profile for the gateway of a smart metering system (Smart Meter Gateway PP), Version 1.3. Bundesamt für Sicherheit in der Informationstechnik.

Buschmann F, Henney K, Schmidt D. Pattern-oriented software architecture, volume 4: a pattern language for distributed computing. Wiley; 2007. ISBN 978–0470059029.

Buschmann G, Meunier R, Rohnert H, Sommerlad P, Stal M, 1. John Wiley and Sons; 1996. ISBN 978–0471958697.

Callas J, Donnerhacke L, Finney H, Shaw D, Thayer R, 2007. OpenPGP message format (RFC 4880). Internet Engineering Task Force.

Cheng BHC, Konrad S, Campbell LA, Wassermann R. Using security patterns to model and analyze security. Proceedings of IEEE workshop on requirements for high assurance systems; 2003. p. 13–22.

Dali A, Lajtha C. Iso 31000 risk management: the gold standard. EDPACS 2012;45(5):1–8.

De Win B, Scandariato R, Buyens K, Grgoire J, Joosen W. On the secure software development process: CLASP, SDL and Touchpoints compared. Inf Softw Technol 2009;51(7):1152–71. doi:10.1016/j.infsof.2008.01.010.

Davis F. Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q 1989;13(3):319.

DECC. Smart metering communications hub, Department of Energy and Climate Change, UK. 2015.

Devanbu P, Stubblebine S, Premkumar SS, Devanbu T. Software engineering for security - a roadmap. Proceedings of the conference on the future of software engineering, ICSE '00. ACM; 2000. p. 227–39.

Dierks T, Rescorla E, 2008. The TLS Protocol Version 1.2 (RFC 5246). Internet Engineering Task Force.

Douglass BP. Real-time UML: developing efficient objects for embedded systems. Addison-Wesley; 1998. ISBN 0-201-32579-9.

Fernandez E. Software design patterns. Security patterns in practice: building secure architectures using software patterns. Wiley; 2013. ISBN 978-1-119-99894-5.

Fernandez E, Yoshioka N, Washizaki H, Jurjens J, VanHilst M, Pernul G. Using security patterns to develop secure systems. Software engineering for secure systems: industrial and research perspectives. IGI Global, 2011.

Fernandez E, Yoshioka N, Washizaki H, VanHilst M. Measuring the level of security introduced by security patterns. Proceedings of international conference on availability, reliability, and security (ARES). IEEE Computer Society; 2010. p. 565–8.

France R, Rumpe B. Model-driven development of complex software: a research roadmap. Future of software engineering (FOSE). IEEE Computer Society; 2007. p. 37–54.

Fuchs A, Gürgens S, Rudolph C. A formal notion of trust – enabling reasoning about security properties. Proceedings of fourth IFIP WG 11.1 international conference on trust management. Springer; 2010. p. 200–15.

Gamma E, Helm R, Johnson RE, JVlissides. Design patterns: elements of reusable object-oriented software. Addison-Wesley; 1995. ISBN 978-0-470-05902-9

Giacomo VD, Felici M, Meduri V, Presenza D, Riccucci C, Tedeschi A. Using security and dependability patterns for reaction processes. Proceedings of the 2008 nineteenth international conference on database and expert systems application. IEEE Computer Society; 2008. p. 315–19.

Gonzalez D, Weber D, 2013. Specification of Platform. Deliverable D6.1 TERESA/WP6/D6.1. IST Project IST-248410, TERESA Consortium.

Grandy H, Haneberg D, Reif W, Stenzel K. Developing provable secure m-commerce applications. Emerging trends in information and communication security. Springer; 2006. p. 115–29.

Gray J, Tolvanen JP, Kelly S, Gokhale A, Neema S, Sprinkle J. Domain-specific modeling. In: Fishwick P, editor. Handbook of dynamic system modeling. Chapman & Hall/CRC; 2007. p. 1–20.

Gürgens S, Ochsenschläger P, Rudolph C. On a formal framework for security properties. Int Comput Stand Interface J (CSI) 2005;27(5):457–66. Special issue on formal methods, techniques and tools for secure and reliable applications.

Halkidis ST, Tsantalis N, Chatzigeorgiou A, Stephanides G. Architectural risk analysis of software systems based on security patterns. IEEE Trans Depend Secur Comput 2008;5(3):129–42.

Howard M. Lessons Learned from Five Years of Building More Secure Software.2007 https://msdn.microsoft.com/en-us/magazine/cc163310.aspx#S1 [Accessed: June 2015].

Hafiz M, Adamczyk P, Johnson RE. Organizing security patterns. IEEE Softw 2007;24:52–60.

Hamid B. A model-driven methodology approach for developing a repository of models. Proceedings of the fourth international conference on model and data engineering - (MEDI). Springer; 2014. p. 29–44.

Hamid B. Interplay of security & dependability and resource using model-driven and pattern-based development. Proceedings of IEEE international conference on trust, security and privacy in computing and communications (TrustCom). IEEE Computer Society; 2015. p. 254–62.

Hamid B. A model-driven approach for developing a model repository: methodology and tool support. Future Generation Computer Systems. Elsevier; 2017. p. 473–90.
Hamid B, Geisel J, Ziani A, Bruel J, Perez J. Model-driven engineering for trusted embedded systems based on security and dependability patterns. Proceedings of sixteenth international SDL forum. Springer; 2013. p. 72–90.
Hamid B, Gürgens S, Fuchs A. Security patterns modeling and formalization for pattern-based development of secure software systems. Innov Syst Softw Eng 2016;12(2):109–40. doi:10.1007/s11334-015-0259-1. Springer.
Hamid B, Gürgens S, Jouvray C, Desnos N. Enforcing S&D pattern design in RCES with modeling and formal approaches. Proceedings of ACM/IEEE international conference on model driven engineering languages and systems (MODELS). Springer; 2011. p. 319–33.
Hamid B, Percebois C, Gouteux D. A methodology for integration of patterns with validation purpose. Proceedings of european conference on pattern language of programs (EuroPlop). ACM DL; 2012. p. 1–14.
Hamid B, Perez J. Supporting Pattern-based dependability engineering via model-driven development: approach, tool-support and empirical validation. J Syst Softw 2016;122:239–73. doi:10.1016/j.jss.2016.09.027. Elsevier
Hatebur D, Heisel M, Schmidt H. A pattern system for security requirements engineering. Proceedings of the second international conference on availability, reliability and security (ARES). IEEE; 2007. p. 356–65.
Hauge A. SaCS: A Method and a Pattern Language for the Development of Conceptual Safety Design. University of Oslo; 2014 [Doctoral thesis]. [Accessed: June 2016].
Hegedüs A, Horváth A, Varró D. A model-driven framework for guided design space exploration. Autom Softw Eng 2015;22(3):399–436.
Henninger S, Corrêa V. Software pattern communities: current practices and challenges. Proceedings of the fourteenth conference on pattern languages of programs. ACM; PLOP '07, 2007. p. 14:1–14:19
IEC, 2010. IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems.
ISO, 2009. ISO 31000 Risk management – Principles and guidelines.
ISO, 2013. ISO 27001 Information technology – Security techniques – Information security management systems – Requirements.
ISO/IEC, 2007. ISO 5408-1 Information technology – Security techniques – Evaluation criteria for IT security – Part 1:Introduction and general model.
Jürjens J, Rumm R. Model-based security analysis of the german health card architecture. Methods Inf Med 2008;47(5):409–16.
Jürjens J. Towards development of secure systems using UMLsec. Fundamental approaches to software engineering. Springer; 2001. p. 187–200.
Jürjens J. Foundations for designing secure architectures. Electron Notes Theor Comput Sci 2006;142:31–46.
Jürjens J, Popp G, Wimmel G. Towards using security patterns in model-based system development. Proceedings of EuroPLoP 2002 conference, 2002.
Katt B, Gander M, Breu R, Felderer M. Enhancing model driven security through pattern refinement techniques. Formal methods for components and objects; 2013. p. 169–83.
Landwehr CE. Formal models for computer security. ACM Comput Surv 1981;13:247–78.
Lee Y, Lee J, Lee Z. Integrating software lifecycle process standards with security engineering. Comput Secur 2002;21(4):345–55.
Lee Y, Lee Z, Lee C. A study of integrating the security engineering process into the software lifecycle process standard (IEEE/EIA 12207). Annual Meeting of the Association for Information Systems Americas Conference on Information Systems (AMCIS), 2009.
Likert R. A technique for the measurement of attitudes. Arch Psychol 1932(140):5–55.
Lodderstedt T, Basin D, Doser J. SecureUML: A UML-Based Modeling Language for Model-Driven Security. Proceedings of the fifth international conference on the unified modeling language. London, UK: Springer-Verlag; UML '02; 2002. p. 426–41.
Lucio L, Zhang Q, Nguyen PH, Amrani M, Klein J, Vangheluwe H, Traon YL. Advances in model-driven security. Adv Comput 2014;93:103–52.
McDonald J, Oualha N, Puccetti A, Hecker A, Planchon F. Application of EBIOS for the risk assessment of ICT use in electrical distribution sub-stations. Proceedings of PowerTech (POWERTECH). IEEE; 2013. p. 1–6.
Microsoft. Microsoft Security Development Lifecycle (SDL) Process Guidance - Version 5.2. 2012. [Accessed: November 2015].
Mikkonen T. Formalizing Design Patterns. Proceedings of the twentieth international conference on software engineering (ICSE). IEEE Computer Society; 1998. p. 115–24.
Moebius N, Stenzel K, Grandy H, Reif W. SecureMDD: a model-driven development method for secure smart card applications. Proceedings of international conference on availability, reliability and security, 2009. ARES'09. IEEE; 2009. p. 841–6.
Neumann P. Principled assuredly trustworthy composable architectures. Technical Report SRI Project P11459; DARPA; 2004.
Noble J. Classifying relationships between object-oriented design patterns. Proceedings of the Australian software engineering conference (ASWEC). IEEE Computer Society; 1998. p. 98–107.
OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT), Version 1.1. 2011. http://www.omg.org/spec/QVT/1.1/ [Accessed: January 2013].
OWASP. Application threat modeling. 2017a. https://www.owasp.org/index.php/Application_Threat_Modeling [Accessed: December 2017].
OWASP. Owasp top ten project. 2017b. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project [Accessed: December 2017].
Paulson L.. Proving Properties of Security Protocols by Induction. Technical Report 409; Computer Laboratory, University of Cambridg; 1996.
Peffers K, Tuunanen T, Rothenberger M, Chatterjee S. A design science research methodology for information systems research. J Manage Inf Syst 2007;24(3):45–77.
Radermacher A, Hamid B, Fredj M, Profizi JL. Process and tool support for design patterns with safety requirements. Proceedings of European conference on pattern language of programs (EuroPlop). ACM DL, 2013. p 8:1–8:16
Rescorla E, Modadugu N, 2012. Datagram Transport Layer Security Version 1.2 (RFC 6347). Internet Engineering Task Force.
Riehle D, Züllighoven H. Understanding and using patterns in software development. Theor Pract Object Syst 1996;2(1):3–13.
Rogers E. Diffusion of innovations. fifth ed. New York, USA: Free Press; 2003.
Runeson P, Höst M. Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 2009;14(2):131–64.
Schmidt H, Jürjens J. Connecting security requirements analysis and secure design using patterns and UMLsec. Proceedings of twenty third international conference on advanced information systems engineering (CAiSE). Springer; 2011. p. 367–82.

Schneier B. Attack trees, modeling security threats. Dr Dobb's Journal 1999 (December 1999).
Schumacher M. Lecture Notes in Computer Science, 2754. Springer; 2003. ISBN 978-3-540-45180-8.
Schumacher M, Fernandez E, Hybertson D, Buschmann F. Security patterns: integrating security and systems engineering. John Wiley & Sons; 2005. ISBN 978-0-470-85884-4.
Selic B. The pragmatics of model-driven development. IEEE Softw 2003;20(5):19–25.
Serrano D, Mana A, Sotirious AD. Towards precise and certified security patterns. Proceedings of second international workshop on secure systems methodologies using patterns (Spattern). IEEE Computer Society; 2008. p. 287–91.
Soundarajan N, Hallstrom J. Responsibilities and rewards: specifying design patterns. Proceedings of the twenty sixth international conference on software engineering. IEEE Computer Society; 2004. p. 666–75.
Srivatanakul T, Clark JA, Polack F. Effective security requirements analysis: HAZOP and use cases. Information Security (ISC). Lecture Notes in Computer Science, vol 3225 of LNCS. Springer; 2004. p. 367–82.
Stine, K., Kissel, R., C. Barker, W., Fahlsing, J., Gulick, J., 2008. Guide for Mapping Types of Information and Information Systems to Security Categories. National Institute of Standards and Technology.
Tomer A, Goldin L, Kuflik T, Kimchi E, Schach S. Evaluating software reuse alternatives: a model and its application to an industrial case study. IEEE Trans Softw Eng 2004;30(9):601–12.
Uzunov AV, Fernandez E, Falkner K. Engineering security into distributed systems: a survey of methodologies. J Univers Comput Sci 2013;18(20):2920–3006.
Weiss M, Mouratidis H. Selecting security patterns that fulfill security requirements. Proceedings of the sixteenth IEEE international requirements engineering conference. IEEE Computer Society; 2008. p. 169–72.
Yoder J, Barcalow J. Architectural patterns for enabling application security. Proceedings of conference on pattern languages of programs (PLoP), 1998.
Zdun U, Avgeriou P. A catalog of architectural primitives for modeling architectural patterns. J Inf Softw Technol 2008;50(9–10):1003–34.

Dr Brahim HAMID is an associate professor at the University of Toulouse Jean-Jaurés (France) and he is a member of the IRIT-ARGOS team. He got his PhD degree in 2007 in the area of dependability from the University of Bordeaux. He has been an assistant professor at ENSEIRB. Then he worked as a post-doc in the modeling group at the CEA. His main research topics are software languages engineering, at both the foundations and application level, particularly for resource constrained systems. He works on security, dependability and software architecture. Furthermore, he is an expert in model-driven development approaches both in research and teaching. He has participated in a number of national and European research projects. In particular, he has led successfully the IRIT effort on the TERESA FP7 European project, and several national projects. Brahim Hamid is author or co-author of over 50 internationally reviewed publications, mostly on software engineering and IT security and dependability, and he has co-organized several international workshops (DANCE, SD4RCES). He serves as a reviewer in numerous leading journals of the software engineering domain (SOSYM, ADHOC networks, JSA, JSS, JSME, SPE, DIST, etc.c), and as a member of various international conference program committees. He has been invited to do expertise work for various organizations: FWF, FIT (Austria) and for various national research programs (ANR, CIR, etc.). He is also participating in several working groups and involved in several teaching activities related to security and system engineering, as well as engaging technology transfer to other organizations and other bodies or agencies, and more generally through consulting and training activities.

Donatus Weber, studied Applied Computer Science Minor Electrical Engineering. He finished in 2007 with a thesis on implementation of control algorithms in embedded systems for unmanned aerial vehicles. During his work at the Microdrones GmbH, a company developing and producing unmanned aerial vehicles with a wide range of usage, he gained a lot of experience on embedded systems engineering. In 2008 he started to work as Scientific Assistant at the Institute for Data Communications Systems. He got his PhD degree in 2013 in the area of Security Engineering for Embedded Systems in Metering. He has participated in a number of national and European research projects. In particular, he has led successfully the University of Siegen effort on the TERESA FP7 European project.