Ingénierie des sytèmes sécurisés : patrons, modèles et analyses

Anas Motii

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *10/11/2017* **par :**

ANAS MOTII

Engineering Secure Software Architectures: Patterns, Models and Analysis

---

JURY

ABDELMALEK BENZEKRI	Professeur d'Université	Membre du Jury
ANTONIO MANÃ	Professeur d'Université	Membre du Jury
JEAN-MICHEL BRUEL	Professeur d'Université	Membre du Jury
LAURENT PAUTET	Professeur d'Université	Membre du Jury
YVES ROUDIER	Professeur d'Université	Membre du Jury
BRAHIM HAMID	Maître de Conférences	Membre du Jury
AGNÈS LANUSSE	Ingénieur Chercheur	Membre du Jury
CLAIRE PAGETTI	Ingénieur Chercheur	Membre du Jury

École doctorale et spécialité :
MITT : Domaine STIC : Sureté de logiciel et calcul de haute performance

Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (UMR 5505)

Directeur(s) de Thèse :
Brahim HAMID et *Agnès LANUSSE*

Rapporteurs :
Antonio MANÃ et *Laurent PAUTET*

Résumé

De nos jours la plupart des organisations pour ne pas dire toutes, dépendent des technologies de l'information et de la communication (TIC) pour supporter plusieurs tâches et processus (quelquefois critiques). Cependant, dans la plupart des cas, les organisations et en particulier les petites entreprises accordent une importance limitée à l'information et à sa sécurité. En outre, sécuriser de tels systèmes est une tâche difficile en raison de la complexité et de la connectivité croissante du matériel et du logiciel dans le développement des TICs. Cet aspect doit alors être pris en compte dès les premières phases de conception. Dans ce travail, nous proposons une approche basée sur les modèles permettant de sécuriser des architectures logicielles en utilisant des patrons. Les contributions de ce travail sont : (1) un cadre de conception intégré pour la spécification et l'analyse d'architectures logicielles sécurisées, (2) une nouvelle méthodologie à base de modèles et de patrons et (3) une suite d'outils.

Le fondement de l'approche associe un environnement basé sur des langages de modélisation pour la spécification et l'analyse des modèles d'architectures sécurisées et un dépôt à base de modèles d'artéfacts dédiés à la sécurité (modèle de patrons de sécurité, menaces et propriétés de sécurités) permettant la réutilisation de savoir-faire et de connaissances capitalisées. Pour cela on utilise des langages de modélisation pour la spécification et l'analyse de l'architecture. Le processus associé est constitué des activités suivantes : (a) analyse de risques à base de modèle appliquée à l'architecture du système pour identifier des menaces, (b) sélection et importation de modèles de patrons de sécurité, afin d'arrêter ou de mitiger les menaces identifiées, vers l'environnement de modélisation cible, (c) intégration de modèles de patrons dans le modèle d'architecture, (d) analyse de l'architecture obtenue par rapports aux exigences non-fonctionnelles et aux menaces résiduelles. Dans ce cadre, on s'est focalisé sur la vérification du maintien du respect des contraintes temporelles après application des patrons. La recherche de menaces résiduelles est réalisée à l'aide de techniques de vérification exploitant une représentation formelle des scénarios de menaces issus du modèle STRIDE et basés sur des référentiels de menaces existants (ex., CAPEC).

Dans le cadre de l'assistance pour le développement des architectures sécurisées, nous avons implémenté une suite structurée d'outils autour du framework SEMCO et de la plateforme Eclipse Papyrus pour supporter les différentes activités basées sur un ensemble de langages de modélisation conforme à des standards OMG (UML et ses profils). Les solutions proposées ont été évaluées à travers l'utilisation d'un cas d'étude autour des systèmes SCADA (systèmes de contrôle et d'acquisition de données).

Abstract

Nowadays most organizations depend on Information and Communication Technologies (ICT) to perform their daily tasks (sometimes highly critical). However, in most cases, organizations and particularly small ones place limited value on information and its security. In the same time, achieving security in such systems is a difficult task because of the increasing complexity and connectivity in ICT development. In addition, security has impacts on many attributes such as openness, safety and usability. Thus, security becomes a very important aspect that should be considered in early phases of development. In this work, we propose an approach in order to secure ICT software architectures during their development by considering the aforementioned issues. The contributions of this work are threefold: (1) an integrated design framework for the specification and analysis of secure software architectures, (2) a novel model- and pattern-based methodology and (3) a set of supporting tools.

The approach associates a modeling environment based on a set of modeling languages for specifying and analyzing architecture models and a reuse model repository of modeling artifacts (security pattern, threat and security property models) which allows reuse of capitalized security related know-how. The approach consists of the following steps: (a) model-based risk assessment performed on the architecture to identify threats, (b) selection and instantiation of security pattern models towards the modeling environment for stopping or mitigating the identified threats, (c) integration of security pattern models into the architecture model, (d) analysis of the produced architecture model with regards to other non-functional requirements and residual threats. In this context, we focus on real-time constraints satisfaction preservation after application of security patterns. Enumerating the residual threats is done by checking techniques over the architecture against formalized threat scenarios from the STRIDE model and based on existing threat references (e.g., CAPEC).

As part of the assistance for the development of secure architectures, we have implemented a tool chain based on SEMCO and Eclipse Papyrus to support the different activities based on a set of modeling languages compliant with OMG standards (UML and

its profiles). The assessment of our work is presented via a SCADA system (Supervisory Control And Data Acquisition) case study.

Keywords: ICT, Security Engineering, Risk Assessment, Pattern-Based System Engineering (PBSE), Security patterns, Model-Driven Engineering (MDE), UML.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisors Dr. Agnès Lanusse and Dr. Brahim Hamid and Prof. Jean-Michel Bruel for their continuous support of my Ph.D. study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis.

Besides my advisors, I would like to thank my thesis reviewers: Prof. Antonio Manã and Prof. Laurent Pautet for their insightful comments and encouragement, but also for the hard questions which incented me to widen my research from various perspectives. I would like also to thank my thesis committee: Prof. Abdelmalek Benzekri, Prof. Yves Roudier and Dr. Claire Pagetti.

My sincere thanks also goes to Dr. Sébastien Gérard who provided me an opportunity to join his team as a Ph.D. student, and who gave me access to the laboratory and research facilities. Without his precious support it would not be possible to conduct this research.

I would like to thank Ansgar and Chokri for their valuable support. I would like to thank all my Ph.D. colleagues for the fun times that we spent together.

Last but not the least, I would like to thank my wife Jamila who encouraged me during the rough times when I lost hope. I could not have imagined how this thesis would have gone without her support. I thank my son Mohammad Malik who, although is only two years old, motivated me when I was back home after a tough day (you are my hero). I thank my parents and my brothers for supporting me spiritually throughout writing this thesis and my life in general

Table of Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Contents

1.1 Context

Recently, our society has become more dependent on software-intensive systems, such as Information and Communication Technologies (ICTs), not only in safety-critical areas but also in areas such as finance, medical information management and systems using web applications. In such areas, protecting information is compulsory because much of the value of a business is concentrated in the value of its information. Threats to information systems from criminals and terrorists are increasing. Therefore, these systems should also satisfy assurance requirements and standards such as ISO 27005 [78]. In the past, security was not as critical because the connectivity was limited and thus protection was based on the principle of *obscurity and isolation* [32]. Nowadays ICTs have grown in terms of functionality, complexity and connectivity. Therefore, security requirements become more important as well as more difficult to achieve. These challenges have lead ICT experts to search for new methods and tools for securing ICTs.

Industry and academia both claim that security should be treated in early stages of the software and systems development life cycle [160]. Otherwise, security vulnerabilities

are more likely to be introduced in various stages [108] and the cost of protecting them becomes increasingly more important. In a recent event [6] in March 2016, a hacktivist cyber-attack impacted a water treatment system by altering flow rates and chemical quantities. A security risk assessment performed on the company's IT systems showed that the hackers exploited a lack of security control such as authentication mechanisms that could have been identified at architecture level. In that sense, information security risk assessment is essential to detect potential architectural vulnerabilities.

Security risk assessment is usually done by a security risk analyst in order to verify the actual status of an information system that is already deployed. Risk assessment is performed by a set of meetings between security experts and persons responsible for the information system. The main goal is to produce a report with actual security risks targeting the system, the security strategy to adopt and the security measures to deploy in order to achieve this strategy. After achieving risk assessment, ICT architects and developers implement and deploy the software security measures.

The first issue here is that developers are constrained by the functional requirements of the information system that can hardly change because it would mean choosing a new software system which is very expensive. In order to solve the first problem, security must be thought at early stages. In our work we focus on the architecture development stage where design decisions are still flexible.

The second issue is related to the fact that ICT architects and developers usually have basic knowledge in security engineering but lack expertise and best practices to apply the correct recommendations issued by security risk assessment (if any). One solution is to use patterns. In fact, capturing and providing this expertise by the way of security patterns has become an area of research in the last years. Security can be captured within patterns that provide reusable generic solutions for recurring security problems, here dealing with architectural problems. Recently a complete catalog of security patterns has been introduced by Fernandez [48].

1.2 Problem statement

Based on the previous discussion, we specify our general research problem coming from the lack of methodological tool support of both the system architecture and security.

The need to address the problem of defining a methodological tool support for the specification and analysis of secure software architectures, which reduces the cost of engineering secure ICT systems.

1.3 Research goals

Taking into account the previous discussion, we specify our research problem as an overall research goal of this thesis:

Define and assess an innovative framework for the development of system architecture and security using patterns. More precisely, the aim of this work is to provide formalisms, techniques, methods and tools allowing a safer secure system architecture development, in the context of future automated security-by-design for the development of ICT systems.

Special emphasis will be to devoted to promote the particularly challenging task of selecting and integrating security solutions within restricted design space of real-time ICT. Furthermore, potential benefits of the combination of Model-Driven Engineering (MDE) with a pattern-based representation of security solutions will be considered and evaluated.

Unfortunately the use of patterns brings two major problems. First, traditional security patterns are usually described as informal guidelines to solve a certain problem using templates such as POSA [29] and GoF [52]. Hence even if security patterns have advantages in fostering reuse, their use is difficult in reality. In fact, there is a major gap of understanding between threats issued by risk assessment and the description of a security pattern against these threats with a template. Furthermore, another difficulty is added during their integration i.e., incorporation into the architecture. Due to manual security pattern integration, the problem of incorrect integration (one of the most important source of security problems) remains unsolved. Second, ICTs generally involve multi-concern objectives. Indeed, systems such as Cyber-Physical Systems (CPS) must satisfy a number of requirements (real-time, physical, energy efficiency and others) where security may have an impact on. Therefore, architects must apply trade-offs to satisfy security requirements and other non-functional requirements. However, even if the use of security patterns has its advantages by fostering reuse, other development life cycle process quality attributes must be guaranteed (e.g., cost, time and efficiency).

These two problems can be solved with Model-Driven Engineering (MDE). In fact MDE provides a useful contribution for the design and analysis of secure systems due to abstraction and generation mechanisms. In addition, it makes easier the enactment of the separation of concern paradigm (security, real-time, performance, etc.). It helps the architect to analyse and to evaluate in a separate view non-functional requirements such as security at a high level of abstraction. We leverage on this idea to propose a tool-supported model-based process for the use of security patterns at the architectural level and the analysis of the architecture solution (obtained after the use of security patterns)

intended for systems with stringent security requirements.

Though there is research dealing with model-based security engineering [85, 99, 82, 100], more research is needed for improving the selection and integration of security patterns based on security requirements and recommendations and the analysis of architecture refinements against other non-functional requirements.

In this work, we present a model-based approach for engineering secure software systems that uses patterns to represent security solutions and knowledge, which fosters reuse. Based on the above, this work has the following top-level research goals:

Research goal 1.

Develop new modeling languages for the specification and analysis of software system architecture and security. **RG1**

Research goal 2.

Develop a methodological holistic approach towards software system architecture and security engineering using patterns. **RG2**

Research goal 3.

Build security and architecture techniques for the analysis of software architecture against security and real-time requirements. **RG3**

Research goal 4.

Build a new operational tool suite to support the proposed approach. **RG4**

Research goal 5.

Study the feasibility of the proposed approach through its application to different sectors. **RG5**

1.4 Contributions

The contributions of this work are threefold: (1) an integrated design framework for the specification and analysis of secure software architectures, (2) a novel model- and pattern-based methodology and (3) a set of supporting tools. Here, we map the contributions of this thesis to the goals formulated earlier.

RG1 is addressed in the following contributions:

1. **EBIOS UML profile.** EBIOS concepts [14] are used to create a UML profile in order o represent risk assessment artifacts.

2. **Architecture specification.** We use the extension mechanisms of UML [130] and create a UML profile for component-based software architectures. The profile is then augmented with verifiable constraints, written in the Object Constraint Language (OCL), that help system architect in systematically identifying security threats.

3. **Pattern specification.** The SEPM metamodel [61] is used to create a UML profile in order o represent security patterns. This UML profile has been developed to allow the instantiation of patterns in the targeted modeling environment.

4. **Integration.** In order support the pattern integration methodology, we create a UML profile that helps architects in relating the concepts of the solution provided by the pattern to those of the target architecture.

RG2 is addressed in the following contributions:

1. **Pattern selection and instantiation.** We propose a method to select and instantiate proper architectural security patterns from a model repository based on risk assessment recommendations.

2. **Pattern integration.** We propose method for pattern integration based on Model-To-Model transformations and constraints checking.

RG3 is addressed in the following contributions:

1. **Architecture threat analysis.** We propose an iterative process for threat identification in secure software architecture solutions against formalized threat scenarios. The process comes in complement to security risk assessment. In fact risk assessment is done on a system architecture. In our context a risk assessment phase is achieved at system architecture level, then threats are reevaluated at each refinement stage in the software architecture design.

2. **Architecture real-time analysis.** We propose a process for analyzing software architecture candidate solutions against real-time constraints.

RG4 is addressed in the following contributions:

1. **EBIOS Risk Analysis.** This module enables the modeling of feared events, threats and risks. It allows the automatic computation of risk level from likelihood and impact levels. These risks can be categorized according to security property and threat categories.

2. **Automatic Threat Analysis.** This module enables architects to evaluate a software architecture solution against formalized attack scenarios.

3. **Access tool.** This module takes the form of a *GUI access* tool where the architect requests a set pattern models providing a number of security properties. Thanks to Model-To-Model transformations, these pattern models are instantiated into the modeling environment.

4. **PatternIntegrator.** This module enables architects to integrate correctly selected security pattern models into the software architecture. This integration is achieved through Model-To-Model transformation and devoted constraints verification using OMG standards (QVTo and OCL).

RG5 is addressed in the following contributions:

- **Feasibility**: the overall methodology and the related tools have been applied on a reference SCADA (Supervisory Control and Data Acquisition) case study [161]. The results are compared with literature results and discussed.

1.5 Publications

This section presents our published papers related to the thesis.

- **Paper A.** [117] *Guiding the selection of security patterns for real-time systems (regular paper).* **Anas Motii**, Brahim Hamid, Agnes Lanusse, Jean-Michel Bruel. : *IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2016)*, IEEE, November 2016.

  Summary: In this paper, we propose a model-based approach for evaluating proper security solution alternatives composed of security patterns at early design stage against real-time requirements. We provide a generalizable and tool-supported solution to support the approach using UML and its profiles. A validation of the work is presented via a simplified version of SCADA (Supervisory Control and Data Acquisition) system case study.

- **Paper B.** [118] *Towards the integration of security patterns in UML component-based applications.* **Anas Motii**, Brahim Hamid, Agnes Lanusse, Jean-Michel Bruel. *PAME workshop (Models 2016)*, Springer, October 2016.

  Summary: This paper is about the use of patterns in secure systems and software engineering, in particular in Model-Driven Engineering. In this paper, we are proposing a model-based process for security pattern integration aimed to secure component-based software system architectures in UML. The methodology is based on merging techniques and verifications against integration constraints described in OCL language. The paper illustrates the method through a Virtual Private Network (VPN) pattern.

- **Paper C.** [119] *Model-Based Real-Time Evaluation of Security Patterns: A SCADA System Case Study.* **Anas Motii**, Agnes Lanusse, Brahim Hamid, Jean-Michel Bruel. *TIPS workshop (SAFECOMP 2016)*, Springer, September 2016.

  Summary: In this paper, a detailed illustration of a SCADA system case study and tool chain, is presented based on Papyrus UML.

- **Paper D.** [116] *Guiding the selection of security patterns based on security requirements and pattern classification.* **Anas Motii**, Brahim Hamid, Agnes Lanusse, Jean-Michel Bruel. *European Pattern Languages of Programs Conference (PLOP 2013)*, ACM DL, July 2015.

  Summary: In this paper, security patterns are selected by developers based on security requirements. On the other hand, security risk management is an iterative approach that consists of: (1) a risk assessment activity for identifying, analyzing and evaluating security risks and (2) a risk treatment activity to mitigate these risks which results in issuing security requirements. Hence, risk management and security PBSE can be used together. In this context, this paper aims at guiding the selection of security patterns in security PBSE based on security risk management results and pattern classification. For illustration purposes, we consider an example of a SCADA (Supervisory Control And Data Acquisition) system.

- **Paper E.** [7] *Using Model Driven Engineering to Support Multi-paradigms Security Analysis (Revised Selected Papers).* Rouwaida Abdallah, **Anas Motii**, Nataliya Yakymets, Agnes Lanusse. *International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2015)*, Springer, 2015

  Summary: In this work, we propose a methodology and associated framework for

security analysis. The methodology relies upon model-driven engineering approach and combines two types of methods: a qualitative method named EBIOS that is usually simple and helps to identify critical parts of the system; then a quantitative method, the Attack Trees method, that is more complex but gives more accurate results. We present the automatic generation of Attack trees from EBIOS analysis phase. We show on a SCADA system case study how our process can be applied.

1.6 Thesis outline

The outline of the thesis dissertation is as follows. Chapter 2 presents the background and context of our work. Case studies are also introduced in this section. Chapter 3 presents a global overview of our approach and its positioning with regards to the software development process. Following are chapters 4, 5 and 6 relevant to the contributions and their positioning with regards to related research. Chapter 4 presents an approach and tool-support for the selection, instantiation and integration of proper security patterns during architecture design based on risk assessment recommendations. Chapters 5 and 6 present the analysis process and its tool support for analyzing architecture solution candidates with regards to real-time requirements and formalized threat scenarios. In chapter 7, the feasibility of the contributions are studied and discussed through a SCADA (Supervisory Control and Data Acquisition) system case study. Finally, chapter 8 concludes the dissertation and proposes some future works and perspectives.

Chapter 2

Context

Contents

2.1 Introduction

The design of ICT is an inherently complex endeavor. In particular, the integration of non-functional requirements such as security is exacerbating this complexity. MDE is a promising approach for the design of trusted systems, since it bridges the gap between design issues and implementation concerns. MDE has the potential to greatly ease recurring activities of architects. MDE supports the architect to resolve in a separate way non-functional requirements such as security at a higher abstraction level.

In this chapter we present the context of our work, including a set of concepts, definitions that might prove useful in understanding our approach, and an introduction to the case studies.

2.2 Software System Development

2.2.1 Software System Engineering

Shortly after the beginning of software development, the way software is developed has been analyzed and guidelines and best practices have evolved over time in the different application domains. Since then, software system engineering has evolved to an engineering domain, having impacts on the different fields of system development, such as development processes and software product life cycles. The IEEE defines in their Standard ISI/IEC/IEEE 24765:2010 [5] Software Engineering as follows: *Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.* In software systems engineering, two principal concepts intervene in the construction of a system, a) the practical use and economic value, being the balance of what the customer wants and what the customer is ready to pay for, b) the correctness, suitability and safety, being the attempt to ensure the correctness and the suitability of resulting product and the absence of safety critical failures. Tackling these two principal concepts are the challenges of the software engineering discipline, which is based and evolves through application of scientific and mathematical knowledge.

2.2.2 Component-Based Engineering

Description

Component models provide a way to cope with some limitations of the object model. Component models complement the object model by providing an architectural view of the application. Therefore component models provide a coarser grain representation of the application: a component is typically implemented as a compound of objects or of other components, therefore providing different levels of abstraction for representing complex systems. The main additions of component models to object models can be summarized as follows:

- Identification of connection points between components and the associated links,
- Identification of the services required by a client (instead of just the services provided by a server)
- New communication patterns (for example event-based communication).

Component technology has become a central focus of software engineering in research and development due to its great success in market. Reuse is a key factor that contributes to this success. The basic idea in component-based software engineering (CBSE) is building systems from existing components rather than "reinventing the wheel" each time. The components are built to be reused in different systems and the component development process is separated from the system development process [154].

Szyperski's definition of a component [150]: *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.* There are several component-based methodologies Catalysis [41], KobrA [16], Fusion (Coleman 1993), OPEN process framework [54]. Flex-eWare [81] is model-driven solution for designing and implementing embedded distributed systems. It combines Model-Driven Engineering and Component-Based Engineering. Bagnato et al. present a framework called EAST-AADL [18] which is an architecture description language for the automotive domain supported with a methodology compliant with the ISO 26262 standard. The language and the methodology set the stage for a high-level of automation and integration of advanced analyses and optimization capabilities to effectively improve development processes of modern cars.

Component-Based Development Process

Component-Based Development (CBD) processes are flexible to many software development processes (e.g., V cycle, Waterfall, Agile, etc.). In our context, the main concern is to achieve architecture design with components rather than foster their reuse [34]. To explain CBD processes, reference development phases corresponding to four abstraction levels of a system model, are identified in Figure 2.1: Requirements, System Architecture Design, Software Architecture Design and Component Internal Design.

In addition, Figure 2.1 shows the different artifacts at each phase (features, analysis functions, design functions and software components) and their m-to-n relationships [18]. Typically, "m-to-n" relationships (from n entities of a higher level to m entities of the lower level) allow refining models throughout the process for an incremental system concretization.

Requirements. During this phase, requirements are analyzed in order to construct a set of features that the system is expected to provide.

System Architecture Design. In this phase, each feature is analyzed and refined with a set of individual function units (e.g., sensing and actuating functions). Hence, the

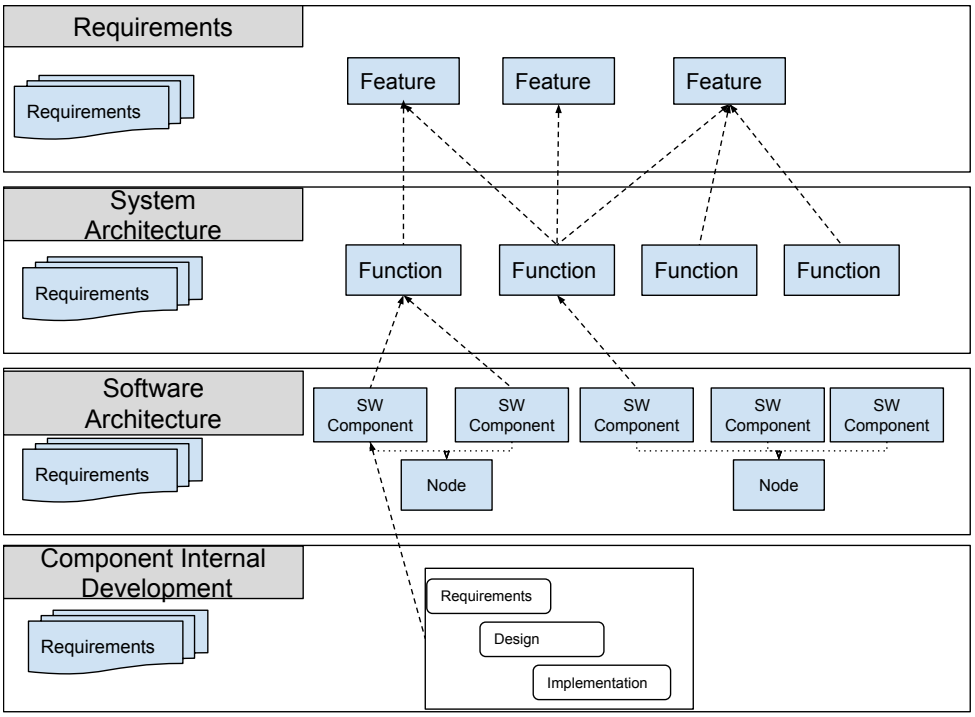


Figure 2.1: Component-Based Software Development Process and Used Artifacts [18]

functions are abstract and independent from any hardware or software.

Software Architecture Design. Each function is refined with a set of software components (which define software components in terms of provided and required interfaces) and hardware nodes.

Component Internal Design. In this phase, each software component is realized. The realization may be done by: (1) combining existing software components or (2) from scratch. Particularly, in CBD methods using UML such as Catalysis [41], each component is designed and implemented internally as a set of classes that implement the interfaces required externally. In addition, internal interactions are realized.

2.2.3 Model-Based Engineering (MBE)

Models are used to denote abstract representation of systems. Especially, we need models to represent software architecture and software platforms to test, to simulate and to validate the proposed solutions. Model-Based Engineering (MBE) solutions seem very promising to meet the needs of secure ICT development. The idea promoted by MBE is to use models at different levels of abstraction for developing systems. In other words, models provide input and output at all stages of system development until the final system

constructed.

Model-Driven Engineering (MDE)

The concept of model is becoming a major paradigm in software engineering. Its use represents a significant advance in terms of level of abstraction, continuity, generality, scalability, etc. Model-Driven Engineering (MDE) is a form of generative engineering [139], in which all or a part of an application is generated from models. MDE is a promising approach since it offers tools to deal with the development of complex systems improving their quality and reducing their development life cycles. The development is based on model approaches, metamodeling, Model-To-Model transformations, development processes and execution platforms. The advantage of having an MDE process is that it clearly defines each step to be taken, forcing the developers to follow a defined methodology. MDE allows to increase software quality and to reduce the software systems development life cycle. Moreover, from a model it is possible to automatize steps by model refinements and generate code for all or parts of the application.

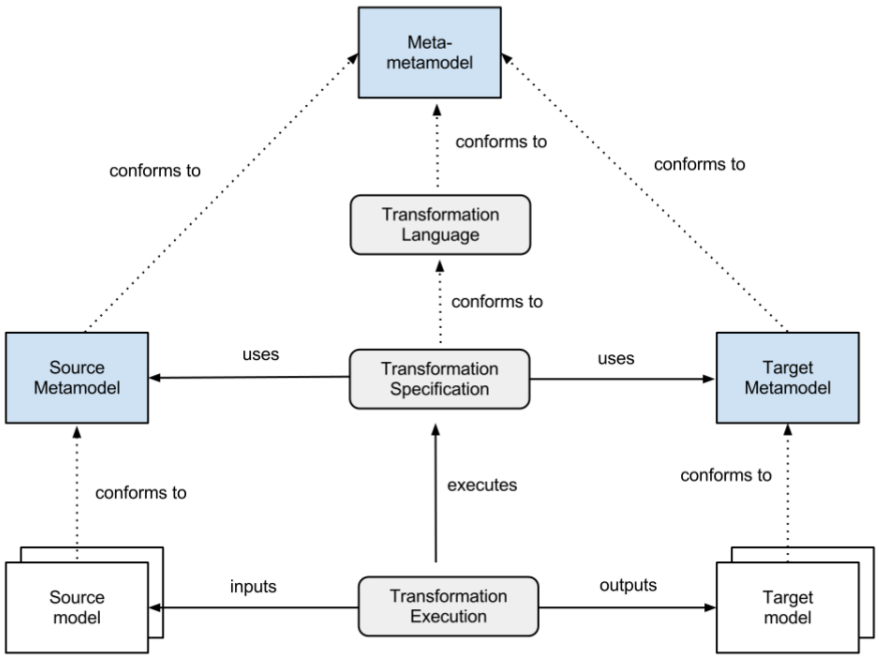


Figure 2.2: MDE: Overview on Model-To-Model Transformation [113]

MDE provides a useful contribution for the design of trusted systems, since it bridges the gap between design issues and implementation concerns. It helps the designer to specify in a separate way non-functional requirements such as security and/or dependability

needs at a higher level of abstraction. This allows implementation independent validation of models, generally considered as an important assurance step.

The development process cycles are mainly iterative, resulting in different levels of model refinement from analysis to design. There are implementation platforms that address these issues in a specific context (e.g. the MDA standard [25]), but in many other contexts, the links between models refined or processed to solve references (to non-existent elements, elements not referenced, created elements, etc.) are still solved in ad-hoc manner, without adequate support from generic technologies.

A Model-To-Model transformation specifies mechanisms to automatically create target models from source models. The Object Management Group (OMG) defines a Model-To-Model transformation as: *the process of converting a model into another model of the same system* [113]. Similarly, [91] defines a Model-To-Model transformation as *the automatic generation of a target model from a source model, according to a transformation description.* Figure 2.2 shows the conceptual background on Model-To-Model transformations as they are used in MDE.

The Meta-Object Facility (MOF) [129] is a standard defined by the OMG to describe modeling languages such as the Unified Modeling Language (UML) [130]. Query View Transformation (QVT) [127], based on the Object Constraint Language (OCL) [126], is an OMG standard to specify Model-To-Model transformations in a formal way, between metamodels conforming to MOF.

Domain Specific Modeling Language (DSML)

A language is defined by an abstract syntax, a concrete syntax and the description of semantics [50, 70, 90]. The abstract syntax defines the concepts and their relationships which is often expressed by a metamodel. The concrete syntax defines the appearance of the language. In this way, a grammar or regular expressions is most of the time used to design this one. On the other hand, semantics define the sense and meaning of the structure by defining sets of rules. Domain Specific Modeling (DSM) is used as a methodology using models as first class citizens to specify applications within a particular domain. The purpose of DSM is to raise the level of abstraction by only using the concepts of the domain and hiding low level implementation details [56]. A Domain Modeling Specific Language (DSML) typically defines concepts and rules of the domain using a metamodel for the abstract syntax, and a concrete syntax (graphical or textual). DSMLs allow to specify systems in a domain-friendly manner. As we shall see, processes in Domain Specific Modeling reuse a lot of practices from Model-Driven Engineering, for

instance, metamodeling and transformation techniques.

DSML: implementation using UML profiles

In order to implement a DSML, there are two methodologies: (1) *Defining all concepts from scratch using a meta-metamodel*, (2) *Using a UML profile*. The first methodology consists on defining the abstract syntax using MOF-based languages such as Ecore using frameworks like Eclipse Modeling Framework (EMF) (see section 2.5.1). The second methodology allows reuse of relevant existing concepts from UML and/or extends them according to specific needs using the UML standardised extension mechanism (UML profiles).

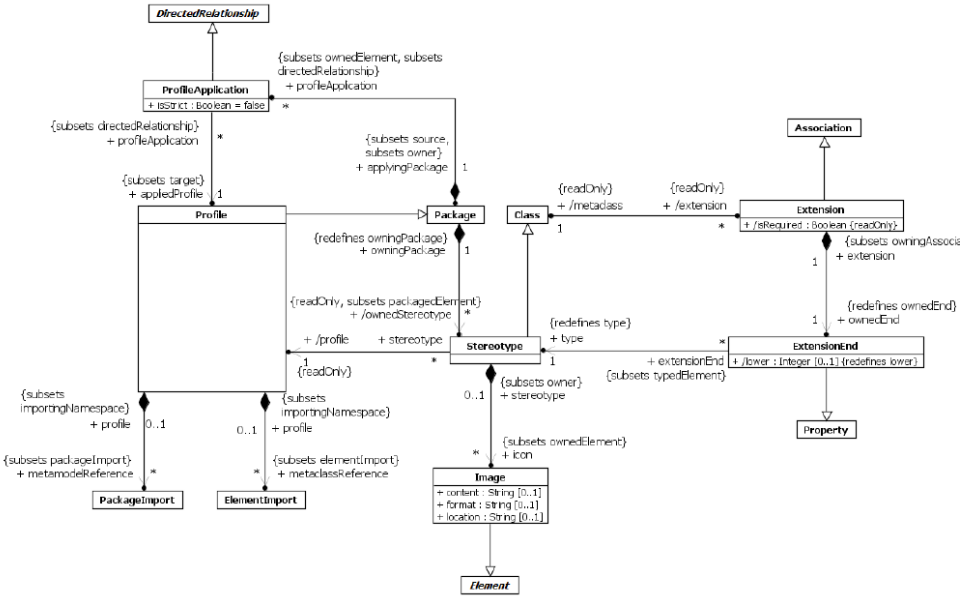


Figure 2.3: Fragment of the UML metamodel for the definition a UML profile

A UML profile [130] is a generic extension mechanism for customizing UML models for specific domains. It is a flexible way to enrich UML models with additional information. Generally, these information cannot be expressed with the use of UML. UML profiles are one way to implement a DSML. The definition of a UML profile consists in extending existing UML concepts, called UML *metaclasses* (e.g., Class, Association, etc.). These extensions are called *stereotypes* and represent concepts of the DSML. The advantage of using such mechanism is to focus only on the definition of the DSML. Another advantage is exploit the existent tools that are based on UML (editors, compilers, simulators) (such as Papyrus, see section 2.5.2).

Constraints, such as OCL constraints, can also be defined in a profile. They can be applied to stereotypes defined in the profile or those imported by the profile. They can also be used to further constrain elements of the UML metamodel. Figure 2.3 shows UML metamodel fragment to consider while defining a UML profile. In our work, we use UML profiles as technological solution for defining DSMLs.

2.3 Incorporating Security in Software Development

In system engineering, security may be compromised on several system layers. Usually, security is considered when design decisions are made leading to potentially conflicting implementations. The integration of security features requires the availability of system architects, application domain specific knowledge and security expertise at the same time to manage the potential consequences of design decisions on the security of a system and on the rest of the architecture. For instance, at architectural level, security means to have a mechanism (it may be a component or integrated into a component). Once a system is engineered, it must be assessed to detect and evaluate risks in order to treat them. Section 2.3.1 presents software systems engineering approaches incorporating security. We mainly elicit the forefront ones: Microsoft SDL, CLASP and Touchpoints. Section 2.3.2 presents risk management, its concepts and the different phases of its process. A special focus is dedicated to the EBIOS methodology.

2.3.1 Software Systems Security Engineering

Approaches taking into account security aspects in software/systems engineering are often considered as Software Systems Security Engineering. In this section we will analyze the state of the art of different approaches in software systems security engineering. On the one hand we will take a look at the integrated approaches taking into account the engineering life cycle from requirements engineering down to software release. In a second section, we will analyze more specific approaches in the Model-Driven Engineering domain, which are in general less holistic and are specialized on different phases, such as requirements engineering or system design.

Generic Software Systems Security Engineering

For the study of existing general purpose security engineering approaches, we limit ourselves to approaches covering a broad spectrum of the development life cycle and proposing

an extensive set of security oriented activities. We will focus on the three forefront representatives, namely Microsoft's Security Development Life cycle (SDL), OWASP's Comprehensive, Lightweight Application Security Process (CLASP) and McGraw's Touchpoints, as they are recognized as the major players in the field. These three secure software development approaches or processes have been extensively validated, either by usage in large-scale development projects [94], reviews by security specialized companies [111, 157] or by being inspired by industrial projects [108]. An overview will also be given on further standards or approaches.

Microsoft SDL. Microsoft's Security Development Life cycle [112] is probably the most rigorous, most tool-supported and more oriented towards large organizations (e.g., Microsoft uses it internally). Microsoft defined this process in 2002 to address security issues frequently faced in development. It contains an extensive set of (security oriented) activities, which can be used as supporting activities in development process models. These activities are often related to functionality-oriented activities and complement them by adding security aspects. Proposed activities are grouped into classical development phases (i.e., Education, Design, Implementation, Verification, Release) to ease the introduction into existing approaches. Vast guidance, such as detailed description of methods and tool support, is available, enabling even less qualified practitioners to achieve the required outcome. These guidances go as far down as to give coding and compiling guidelines, which do not map to process model activities anymore.

CLASP. The Comprehensive, Lightweight Application Security Process (CLASP) [133] by the OWASP Consortium is a lightweight process containing 24 main security activities. It can be customized to fit different projects (activities can be integrated) and focuses on security as the central role of the system. The main focus of Comprehensive, Lightweight Application Security Process (CLASP) is to support engineering processes in which security takes a central role. For this approach the foundations of a secure system are built in the architectural design and though focus is given on this part of the process model. The activities proposed are developed to cover a wide range of security aspects and are conceived from a security-theoretical perspective and defined in an independent manner to allow process designer, wishing to integrate them, a large field of flexibility. Recommendations are given on how to integrate these activities in an existing process, but there is no direct mapping and the coordination is less direct than in other approaches (such as in SDL or Touchpoints). CLASP also offers a rich set of security support resources, such as an extensive list of security vulnerabilities which can be used at different checkpoints throughout the process. The drawbacks of the approach defined by the OWASP consortium are mainly some activity descriptions, although crucial for se-

cure software development, fail to give detailed methodological indications, and the lack of work product descriptions for the proposed activities.

Touchpoints. McGraw's work [107] is based on industrial experience and has been validated over time. It provides a set of best practices regrouped into 7 so-called touchpoints. These touchpoints express the interactions among process developers and security and how the developer can take into account the security aspects by using the framework (e.g., Risk Management, Attack Analysis, Code Review but also Examples and Basic Security Knowledge). The activities focus on risk management and flexibility and offer white-hat and black-hat approaches to increase security. For McGraw, risk management is of elemental importance in software security. The approach tries to enable this by providing a risk management framework supporting the security activities. In [84] McGraw offers, in addition to an extensive set of security knowledge and links to resources, a rich set of examples on security analysis activities and solutions. In giving general guidelines and adaptive activities, the approach can be tailored to most existing software processes focusing on the touchpoints of the existing process and the proposed security enhancements.

ISO/IEC 27001:2005/27034:2011 ISO/IEC 27000 is a family of standards [80], organizations in security tend to implement. It is based on the PDCA (Plan-Do-Check-Act) philosophy. Ideally, with a process approach following ISO/IEC 27000, not only are requirements defined, but the processes to fulfill the requirements are defined as well. ISO/IEC 27001 Information Security Management System (ISMS) requirements standard and ISO/IEC 27034 are providing guidelines for application security and take explicitly a process approach. Like other ISO standards, certification is possible.

Model-Based Software Systems Security Engineering

Model-based security engineering approaches tackle security aspects at different phases of the development. From the organizational context over requirements engineering down to system design and implementation different independent approaches exist. Several approaches have been proposed in literature dealing with security engineering in the requirements phase. Using abuse frames to model and develop the constraints of security requirements on functional requirements and trust assumptions is proposed in [59, 60], allowing the extension of problem frames to determine security requirements. This allows defining security requirements as constraints on functional requirements and trust assumptions. Another approach for security-oriented requirements engineering is proposed by extending use cases to misuse cases [132, 145] to elaborate security threat identifica-

tion. The idea behind this approach is to describe functions the system should not allow, eliciting security requirements and the following constraints on assets. System design model-driven software engineering processes use UML profiles such as SecureMDD [115], SecureUML [99] and UMLsec [84, 85] providing formal specifications for verification of security-oriented systems. SecureUML provides a UML profile based on Role-Based Access Control (RBAC) allowing specifying 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 which allows generating platform-specific models (e.g., JavaCard) from a high-level stereotyped UML model. In addition to guidelines in modeling security aspects, the framework offers verification based on a formal approach on the produced models [55]. UMLsec is a UML profile aiming to support modeling of security-critical systems. The profile allows expressing security relevant information within the existing model and diagrams and thus taking security aspect into account in the overall system development. In addition to approaches focusing on one phase of the development life cycle, one notable holistic MDE approach is given in literature. In [96, 95], the authors propose an integration model for integrating security engineering approaches into software life cycle standards, mapping the concepts of the software life cycle (Institute of Electrical and Electronics Engineers (IEEE) 12207) to security engineering concepts (a set of concepts collected from various security engineering approaches [96]). The approach tries to give an understanding to stakeholders where and when security activities intervene and interact with standard process life cycle activities.

2.3.2 Security Risk Management

Security Risk Management Methods

There are several risk management methods which can be broken down mainly into two essential types: qualitative and quantitative. Quantitative approaches use numbers to rate the probability of an event and its impact. Hence, the risk related to an event is the multiplication of these numbers. The issue with this model is that using numbers is unreliable because probability is not accurate in general. Some quantitative approaches are CORA, ISRAM and Attack trees. Qualitative approaches rate the magnitude of the potential impact of a threat as threshold level (high, low or medium). Some qualitative approaches are: EBIOS [14], CORAS [39], OCTAVE [31], Information System Security Risk Management method (ISSRM) [42] and NIST SP 800-30 [83].

Model-Based Risk Management methods

Authors of [39] propose a framework (CORAS) by offering a UML profile for risk management and system development based on the Unified Process approach. The model-based methodology integrates several techniques to create a complete report of the system risks, gathering all the necessary documentation and threats diagrams. The Microsoft Threat Analysis & Modeling framework [114] allows non-security domain experts to produce a threat model using already known information including business requirements and application architectures. The framework supports automatic threat detection and also produces valuable security features such as: data access control matrix, component access control matrix, subject-object matrix, data flow, call flow, trust flow attack surface and security reports. The RACOMAT framework [57] combines risk assessment and security testing in both ways. In this context Test-Based Risk Assessment (TBRA) improves security testing results and Risk-Based Security Testing improves risk assessment results.

Definitions

Figure 2.4 shows a simplified conceptual model of risk management. For presenting the concepts, we choose the Information System Security Risk Management method (ISSRM) which is in compliance with the ISO/IEC 27005 standard.

The definitions of these concepts are based on [42].

Asset related definitions:

Definition 1 (Asset). *An asset is something that has value in an organization and is essential to reach its objectives. This concept generalizes the business asset and the IS (Information System) asset.*

Definition 2 (Business Asset). *A business asset is an information, process or skill inherent to the business of the organization. It is essential for achieving its objectives (e.g., technical plan, structure calculation process, architectural competence, etc.). Business assets are immaterial.*

Definition 3 (IS Asset). *An IS asset is a component or part of an IS that has value to the organization and is necessary for reaching its goals. IS assets are classified according to these categories: hardware, software, networks, people or facilities involved in the IS security. IS assets are material with the exception of software.*

Definition 4 (Security property). *A security property is a constraint at organization level over business assets that reflects their security needs (e.g., confidentiality, integrity, availability, non-repudiation, accountability etc.). Security properties are used to express security objectives of an IS (e.g., confidentiality of client data) that must be*

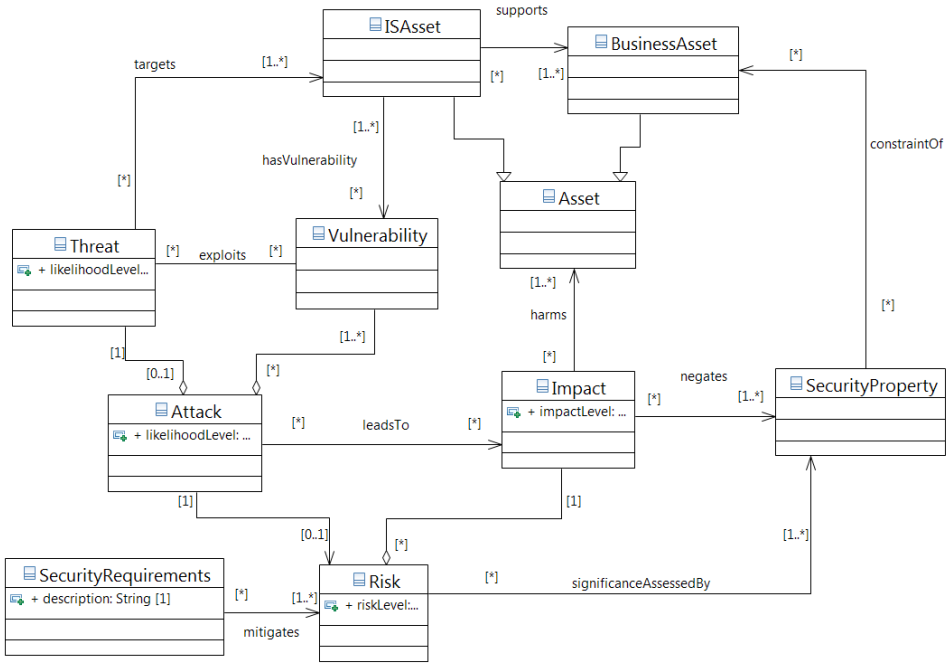


Figure 2.4: Simplified Risk management conceptual model

guaranteed (security requirements)..

Risk related definitions:

Definition 5 (Vulnerability). *A vulnerability is flaw in an IS asset that constitutes a weakness.*

Definition 6 (Threat). *A threat is a potential attack by a threat agent using an attack method. It has a likelihood level.*

Definition 7 (Attack). *An attack is the combination of a threat and one or more vulnerabilities. It has a likelihood level, which is the maximum level among all threats. For example: a hacker uses social engineering on an inexperienced employee exploiting weak awareness.*

Definition 8 (Impact). *A impact is a negative consequence of a risk that harms an asset, when a threat or an attack is accomplished. It has an impact level.*

Definition 9 (Risk). *A risk is the combination of threats with one or more vulnerabilities leading to an impact that harms one or more assets. The risk level is computed as the product of the consequence of impact times the likelihood of the threats.*

Definition 10 (Security requirement). *Security requirements are properties that the system must possess over the environment by installing the IS, in order to mitigate risks.*

ISO/IEC 27005 Process

The ISO 27005 [78] standard belongs to the family of ISO 27000 [80] standards. It provides guidelines for security risk management and supports the concepts defined in ISO 27001 [79] which specifies requirement for the establishment of an Information Security Management System (ISMS). The standard does not provide a particular methodology. However methodologies that claim to perform information security risk management may conform to security standards such as the ISO 27005. The process model of this risk management approach is displayed in Figure 2.5.

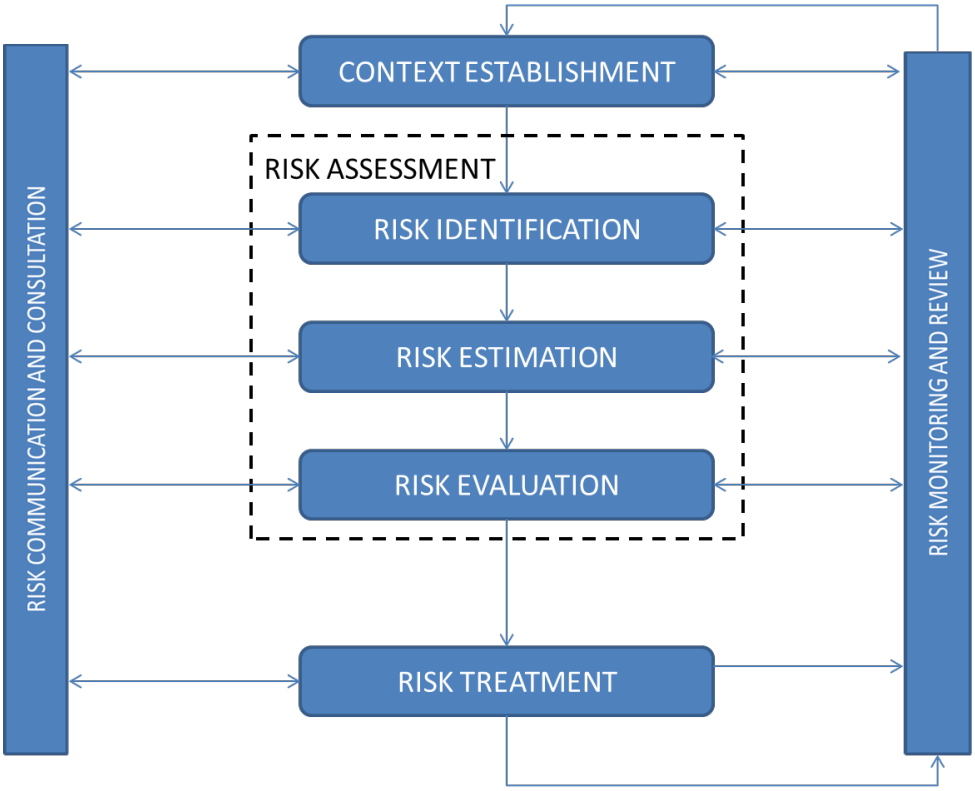


Figure 2.5: ISO 27005 risk management process model [79]

The information security risk management process consists of three main phases:

- **Context establishment.** The aim of this phase is to describe the system under study and to put boundaries to risk management. The system is described in terms of *business assets* and *IT assets* supporting them and the current security controls.

- **Risk assessment.** This phase consists of:

  - *Risk identification:* aims at identifying feared events targeting assets and threats to the IT assets supporting them. A risk is a combination of feared event and a set of threats. There are several ways to enumerate threats e.g., attack trees [140], misuse cases [145] , misuse [27] activities, etc.
  - *Risk estimation:* aims at assigning a likelihood level (occurrence) and impact level (consequence) to each risk by security risk analysts. The assignment of likelihood considers factors such as threat source motivation and capability, nature of the vulnerability and existence and effectiveness of current controls [149]. In the same way, impact is determined by performing an impact analysis [149] that requires information related to the system mission (e.g., the process performed by an IT system expert), system and data criticality (e.g., the system's value or importance to an organization), and system and data sensitivity.
  - *Risk evaluation:* evaluates the risk level that is computed starting from the likelihood and impact levels according to a risk matrix.

- **Risk treatment.** This phase aims at reducing, avoiding, accepting or transferring risks. The output of this activity is a set of security measures.

Risk communication and monitoring are processes that communicate risk information to all decision makers and ensure monitoring of risk levels respectively.

EBIOS method

EBIOS [14] is a security risk management method dedicated to manage risks in information systems operating in steady environments. EBIOS is used by many organizations in both public and private sectors to perform Information System Security (ISS) risk analyzes.

EBIOS method provides uniform vocabulary and concepts that allows attending security objectives. It can be adapted to the context of each organization (its tools and methodologies) and then used to develop either a complete global study of the information system or a detailed study of a particular system. EBIOS consists of five modules as shown in Figure 2.6:

- **Phase 1** deals with context analysis. It establishes the environment, purpose and operation of the target system and identifies the essential (assets) on which they are based.

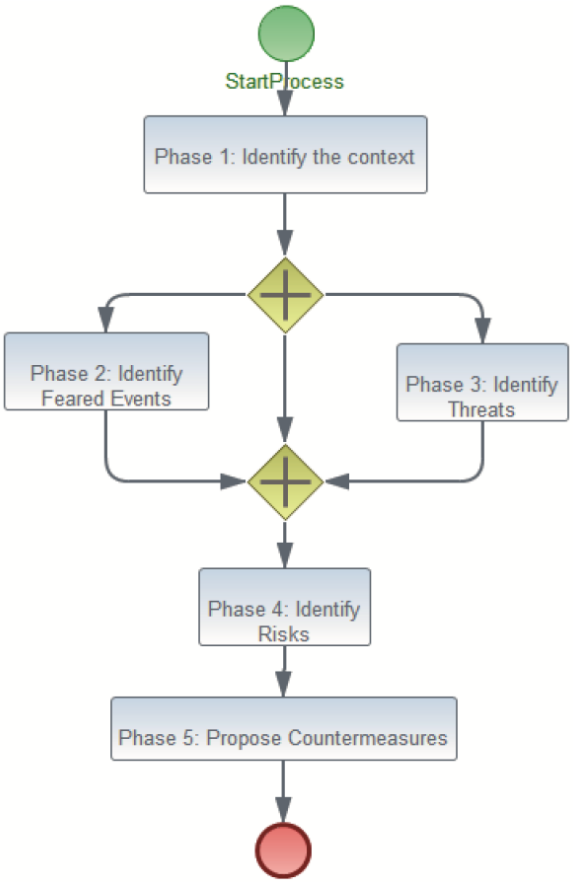


Figure 2.6: EBIOS analysis method

- **Phase 2** conducts the security needs analysis. The identified security needs of the essential elements are evaluated in terms of availability, integrity and confidentiality, etc. In other terms we identify the feared events of the system. We also define a security level to the feared event based on the harm that it may induce.

- **Phase 3** consists of identifying and describing the threats affecting the system. We identify their origin called threat sources, and the vulnerabilities of the systems' elements that can be exploited to apply each attack method. We associate to each threat the likelihood of occurrence.

- **Phase 4** contributes to risk evaluation and treatment. It formalizes the real risks affecting the system by combining threats and feared events. In addition, each risk is treated with a certain strategy (reduce, avoid, transfer or accept).

- **Phase 5** determines how to specify security countermeasures allowing the security

objectives to be fulfilled and how to validate these measures and the residual risk. Actually, for each feared event we associate a risk level. This level is computed based on the likelihood level of the threats and the impact level of the feared event. The risk level is deduced from a predefined matrix. A residual risk is the risk level computed after application of existing or new countermeasures that may decrease the risk level.

Positioning

In our work, we use EBIOS method in order to perform risk assessment. The risk treatment phase is done with the use of security patterns. The goal is to have of repository of patterns and provide facilities to software architects to select and integrate patterns in their environment. In this context, Model-Based Engineering (MBE) provides several techniques to facilitate these tasks.

2.4 Incorporating Security Patterns in Software Development

Pattern-Based System and Software Engineering (PBSE) is a discipline that fosters reuse during system and software development. In our context, PBSE is used during the design of secure ICT software architectures.

2.4.1 Software and Security Patterns

Patterns were introduced by Christopher Alexander [8] in the context of urban planing and building architecture. Since then patterns became popular in the software community. In fact, Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, the Gang of Four (GoF), published the book *Design Patterns - Elements of reusable Object-Oriented Software* [52] about a compilation of design patterns in software engineering. Another book *Pattern Oriented Software Architecture - A System of Patterns* [29] (also known as the POSA book) was published by Buschmann.

Christopher Alexander defined patterns in [8] as follows: *Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to that problem.* In the context of software engineering, Bushmann in the POSA book [29], subdivided patterns into three pattern categories according to the software development process: architecture definition, design and implementation.

- *Architectural patterns* are relevant to the software architecture. They describe: subsystems or components, their interfaces and relationships.
- *Design patterns* refine components and relationships of a software system. They are independent of any programming language.
- *Idioms* are code related patterns and contain best practices for implementation. They are built upon the features of a programming language

In order to describe a security pattern and a pattern in general, a template is needed. A template consists of three fundamental sections: context, problem and solution. There are other sections that add more details about the pattern (e.g., related patterns). There are different templates to describe patterns in software engineering. The most notable ones are POSA and GoF templates. The two templates are similar as they share the same structure. The GoF and POSA differ in classifying the patterns in a catalog. The typical structure of a security pattern is as follow:

- Name: is the name of the security pattern
- Context: describes the environment where the security pattern may be used
- Problem: describes the recurring security problem and the goals and security objectives that pattern wants to reach.
- Forces: define the types of trade-offs that must be considered in the presence of conflicts they might create (e.g., security vs usability)
- Solution: are techniques, structures and mechanisms to solve the problem.
- Related patterns: are relationships with other patterns.

Next to software patterns, several works were interested in security patterns. In 1997, Yoder and Barcalow[163] were the first to work on security pattern documentations. In 2013, Fernandez [48] published a complete book about software security patterns with a methodology to use them through the software development process. Fernandez defines security patterns in [48] as follows: *A security pattern provides a solution to protect a system against a specific threat.* There have been several arguments on the nature of security patterns with regards to the development process. Some authors consider that security patterns are design patterns as in some cases they take the form of the aspect of a subsystem. Some authors considered security patterns as a type of architectural patterns

because they provide solution to software architecture. In our view, security is a global property hence we share the idea that security patterns are architectural.

In order to benefit from all these patterns, pattern catalogs may reveal very useful. Pattern catalogs consist of individual security patterns. The issue behind constructing a catalog of patterns resides in finding out the relationships between them. One challenge is that patterns can follow different templates so incorporating them in a single catalog requires structuring and organizing their sections. Another challenge is the classification of patterns. POSA and GoF propose two different classification approaches. GoF patterns are classified according to three categories: creational, structural, and behavioral. Hence, it is not extensible. It cannot describe other patterns ("architectural patterns") or low level patterns ("idioms"). POSA template is extensible because it classifies each pattern according to its categories and development process. Hence new categories can be added in the classification.

2.4.2 Pattern-Based System engineering

Pattern-Based System Engineering (PBSE) is a system and software engineering discipline that focuses on patterns as its first class citizen. PBSE is similar to Component-Based Software Engineering (CBSE) when it comes to using components but differs in the fact that patterns are not ready-to-use artifacts. This is because patterns are generic and guide towards a solution.

In the context of security patterns, several methods proposed security PBSE methods. In [74], 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 [141] present a detailed description of patterns used in security engineering in different domains. 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 [64], the authors 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 MBE life cycle.

Recently, authors of [67] propose an approach for supporting pattern-based dependability engineering via Model-Driven Development based on the reuse of dedicated subsystems, i.e., so-called dependability patterns that have been pre-engineered to adapt to a specific domain. Model-Driven Engineering (MDE) is used to describe dependability patterns and a methodology for developing dependable software systems using these patterns is proposed.

PBSE must use a catalogue of patterns. Generally catalogues are books, websites, etc. In this case, the development is error-prone and time-consuming which is the opposite of the philosophy of using patterns. PBSE can be coupled with a model repository to benefit from the advantages of MDE.

2.4.3 Patterns in Model Repositories

In the context of MDE, patterns are models. Instead of using a catalog of patterns, patterns are stored in model repositories to comprehensibly explain their classification [63]. A model repository should provide a modeling container to support modeling artifacts associated with different methodologies.

The storage of patterns in a model repository allows to discover the relationships among them and to facilitate the selection of the most appropriate ones. The repository should have a structure in order to optimize the accesses (selecting patterns with criterion's and publishing new patterns into it). Finding the appropriate pattern to solve a particular security or/and dependability problem is difficult because of the lack of a scientific classification scheme for security patterns.

2.5 Tooling

2.5.1 Eclipse Modeling Framework tools

There are several DSM environments, one of them being the open-source Eclipse Modeling Framework (EMF) [148]. EMF provides an implementation of EMOF (Essential MOF), a subset of MOF, called Ecore2. EMF offers a set of tools to specify metamodels in Ecore and to generate other representations of them, for instance Java. In our context, we use the Eclipse Modeling Framework. Note, however, that our vision is not limited to the EMF platform. Here, we outline the different Eclipse tools used in the development of the DSLs to support the modeling of the Security and Dependability (S&D) artifacts, the repository and its APIs. Among the tools used here are cited:

- Eclipse is an open-source software project providing a highly integrated tool platform. The applications in Eclipse are implemented in Java and target many operating systems including Windows, Mac OSX, and Linux [148].
- EMF is a modeling framework and code generation facility for building applications based on a structured data model. In addition, EMF provides the foundation for interoperability with other EMF-based tools and applications [148].
- QVT-O (QVT Operational) [44] allows the implementation of Model-To-Model transformation in Eclipse.
- Acceleo [124] allows the implementation of Model-To-Text transformations in Eclipse.
- CDO (Connected Data Objects) Model Repository is a 3-tier distributed shared model framework for EMF models and metamodels [43].
- RCP plug-in allows developers to use the Eclipse platform to create flexible and extensible desktop applications upon a plug-in architecture [158, 106].

2.5.2 Modeling and Analysis Environment: Papyrus

Papyrus[1] is built on Eclipse Modeling framework (EMF) and offers an implementation of the OMG (Object Management Group) specification Unified Modeling Language (UML) [130]. Papyrus is a UML modeling environment, where many diagrams can be used to view different aspects of a system. Behind all diagrams, there is a model where all modeling elements, used in these diagrams, are kept. The model keeps the consistency between the diagrams. UML diagrams can help system architects and developers understand, collaborate and develop a system. Architects and managers can use diagrams to visualize an entire system or project and separate systems into smaller components for development. System developers can use diagrams to specify, visualize and document systems, which can increase efficiency and improve their system design. Also code can be generated from UML models. Since UML is general-purpose modeling language in the field of software engineering, it is possible to adapt UML to specific domains. This is done by creating and applying UML profiles. Papyrus offers a complete UML modeling environment, which can be used to develop UML profiles. It offers also SysML [125], MARTE [128] and other dedicated profiles implementing DSMLs. In the context of Component-Based Development (CBD), Papyrus supports UCM [131].

[1] https://eclipse.org/papyrus/

2.5.3 Pattern-Based Development Environment: SEMCO

The development of SEMCO (System and software Engineering for embedded systems applications with Multi-COncerns) was started in 2010 by B. Hamid [142]. The SEMCO foundation is a reuse repository of models and patterns and thereby supports a pattern-based development methodology. In SEMCO, a pattern is a subsystem dedicated to non-functional aspects, to be specified by a non-functional aspects experts, and reused by domain engineers to improve systems/software engineering facing non-functional requirements. It is a good application and promotion of Model-Driven Engineering. The core of SEMCO is a set of Domain Specific Modeling Languages (DSMLs), search engines and transformations. The DSMLs are devoted to specify patterns, a System of Patterns as a set of models to govern their use, and thereby to organize, analyze, evaluate and finally validate the potential for reuse. Engines allow to find/select these artifacts from a repository and then transform the results towards specific domain development environments such as Unified Modeling Language (UML).

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 [2] and a CDO-based [3] repository and is currently provided in the form of Eclipse plugins [4].

Definitions

To allow common understanding in the context of the SEMCO framework, we recall a set of definitions elaborated in [164]:

Definition 11 (Domain). *We define a domain as a field or a scope of knowledge or activity that is characterized by the concerns, methods, mechanisms, ... employed in the development of a system. The actual clustering into domains depends on the given group/community implementing the target methodology.*

Definition 12 (Modeling artifact). *We define a modeling artifact as a formalized piece of knowledge for understanding and communicating ideas produced and/or consumed during certain activities of the system engineering processes. The modeling artifact may be classified in accordance with engineering process levels.*

Definition 13 (Modeling artifact system). *A modeling artifact system is a collection of modeling artifacts forming a vocabulary. Such a collection may be skillfully woven*

[2] https://eclipse.org/modeling/emft/
[3] https://eclipse.org/cdo/
[4] http://www.semcomdt.org/semco/tools/updates/1.2

together into a cohesive "whole" that reveals the inherent structures and relationships of its constituent parts toward fulfilling a shared objective.

Definition 14 (Resource). *We define a Resource as a modeling artifact which represents a piece of the platform.*

Definition 15 (Platform). *The platform is defined as a set of interconnected hardware resources on which the software elements can be deployed.*

Definition 16 (Security Pattern). *A security pattern is an architectural pattern that describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution. In our context, a pattern can be abstract or concrete.*

- *Abstract Patterns*: are used at System Architecture level.
- *Concrete Patterns*: are used at Software Architecture level.

Definition 17 (S&D Pattern). *A Security and Dependability (S&D) pattern describes a particular recurring security and/or dependability problem that arises in specific contexts and presents a well-proven generic scheme for its solution.*

Definition 18 (System of Patterns). *We define a System of Patterns as a system of modeling artifacts where its constituent parts are patterns (abstract and concrete), its referenced property models and their relationships.*

Definition 19 (System of Patterns configuration). *A System of Patterns configuration is one of the possible configurations derived from the System of Patterns. It represents the structure of an application based on patterns which will be applied on the application.*

Definition 20 (Repository). *A repository is a shared knowledge base of information on engineered artifacts. They introduce the fact that a repository has (1) a Manager for modeling, retrieving, and managing the objects in a repository, (2) a Database to store the data and (3) features to interact with the repository.*

Definition 21 (Pattern Selection). *An instantiation activity takes a pattern and its related artifacts from the repository and adds it to the end-developer environment. This task enables the pattern to be used while modeling.*

The Selection activity is composed of the following steps:

1. Define needs in terms of properties and/or keywords.
2. Search a pattern in the repository.
3. Identify the appropriate pattern from those proposed by the repository.

Definition 22 (Pattern Instantiation). *An instantiation activity enables the import of the selected pattern(s) into the development environment using Model-To-Model transformation techniques.*

Definition 23 (Pattern Integration). *An integration activity happens within the development environment when a pattern and its related artifacts are introduced into an application design. Some development environments may come with native support for the integration.*

SEMCO

SEMCO is a PBSE approach. It is a federated modeling framework built on an integrated repository of metamodels to deal with system engineering. The end-user part of such a framework is an integrated repository of modeling artifacts to be used in order (i) to model several concerns of embedded systems engineering: extra functional properties; (ii) to model systems parts: logical software, hardware components and infrastructure. The main goal of SEMCO is to deal with multi-concerns embedded system engineering for several domains. PBSE in SEMCO identifies two kinds of processes that interact with the aforementioned model repository:

- *Pattern Development.* concerns designing pattern models from expertise for reuse and storing them in the repository.
- *System Development with Patterns.* concerns selecting appropriate pattern models from the repository with regards to the system-under-development's requirements.

A theory wa built and novel methods based on a repository of models which (1) promote engineering separation of concerns, (2) supports multi-concerns, (3) use model libraries to embed solutions of engineering concerns and (4) supports multi-domain specific process. This framework is threefold: providing repository of modeling artifacts, tools to manage these artifacts and guidelines to build methodologies for system engineering.

As shown in Figure 2.7, SEMCO foundation is a federated DSL processes working as a group on how relevant each one is to the key concern. A DSL building process 2 is divided into several kinds of activities: DSL definition, transformation, consistency and relationships rules, design with DSL and Qualification. The three first activities are achieved by the DSL designer and the two last activities are used by the final DSL user. Figure 2.8 illustrates the use of the Eclipse Modeling Framework (EMF) to support the SEMCO process to create our tool suite.

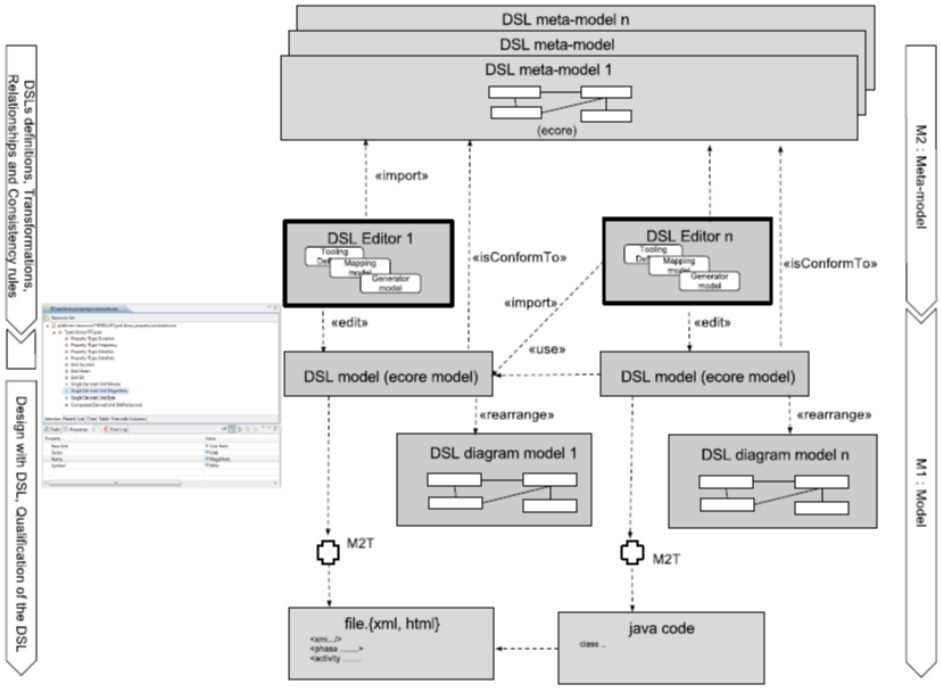


Figure 2.7: Overview of the SEMCO tool suite architecture

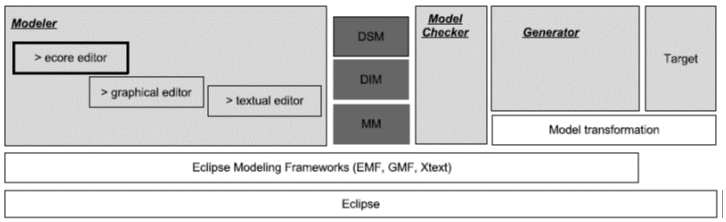


Figure 2.8: SEMCO DSL building process and artifacts

SEPM: A Metamodel for S&D patterns

The System and Software Engineering Pattern Metamodel (SEPM) [61] is a metamodel for describing Security and Dependability (S&D) patterns, and constitutes the base of our pattern modeling language. Here we consider patterns as sub-systems that expose services (via interfaces) and manage S&D and Resource properties (via features) yielding a unified way to capture meta-information related to a pattern and its context of use. The following paragraph details the principal concepts of the SEPM metamodel to specify an S&D pattern, as described with Ecore notations. Figure 2.9 shows the core concepts of SEPM:

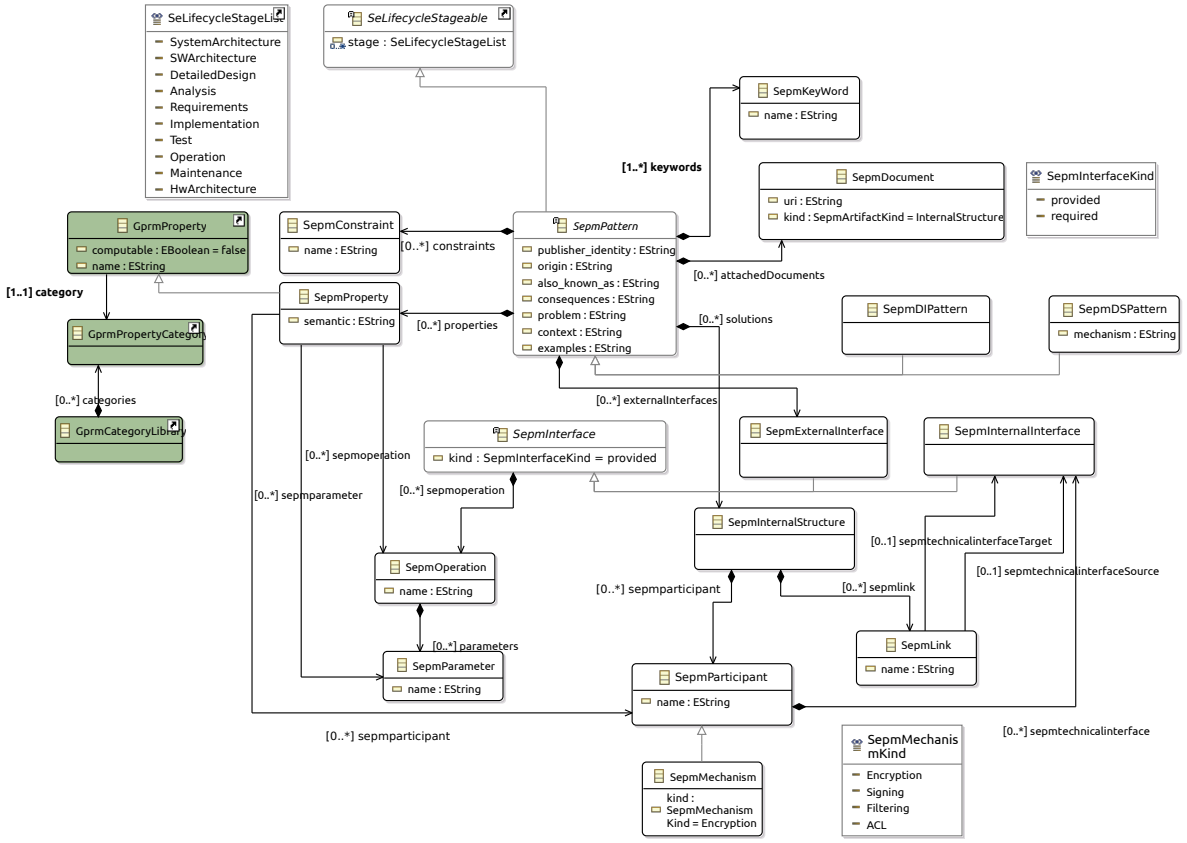


Figure 2.9: The (simplified) SEPM Metamodel

- *SepmPattern.* This concept represents a security pattern as a subsystem describing a solution for a particular recurring design problem that arises in specific design context. It can be either at Domain Independent (DI) or Domain Specific (DS) levels.

- *SepmDIPattern. (abstract pattern)* It inherits from Pattern. A SepmDIPattern is concrete pattern that may be manifested by one or more artifacts, and in turn, that artifact may be deployed by its execution environment. This is the key entry artifact to model pattern at DI level. A DIPattern can be refined from another DIPattern. We call this relation: DILevelRefinement.

- *SepmDSPattern. (concrete pattern)* It inherits from SepmPattern. It is a concrete pattern used to build a pattern at Domain Specific (DS) level. Furthermore a DSPattern has Internal Interfaces in order to interact with the domain specific platform. This is the key entry artifact to model pattern at domain specific level. On one hand, a DSPattern can be built starting from an existing DIPattern. We

call this relation: DomainRefinement. On another hand, a DSPattern can be refined from another DSPattern. We call this relation: DSLevelRefinement.

- *SepmExternalInterface.* A *SepmPattern* interacts with its environment with *Interfaces.*
- *SepmProperty.* A property denotes a particular characteristic of a pattern related to the concern it is dealing with and dedicated to capture its intent in a certain way (e.g., security, dependability and resource properties).
- *SepmInternStructure.* This concept constitutes the implementation of the solution proposed by the pattern. It is composed of SempParticipant and SepmLink.
- *SepmInternalInterface.* It represents an internal interface composed of a set of operations.
- *SepmOperation.* It represents an operation.
- *SepmParameter.* It represents data that is processed by an operation.
- *SepmLink.* It constitutes the basic link of the Static Structure. It is the connection between two Entities.
- *SepmParticipant.* This concept is the main constituent of the solution. It requires and offers a set of interfaces (SepmInternalInterfaces). A participant represents a role that shall be played by an element of the target application. This element is replaced by the participant as a result of pattern integration.
- *SepmMechanism.* is a SepmParticipant with a security-specific purpose (encryption, decryption, signing, etc.). Its role is to add new functionality to the system that is specific to a security property the system should uphold.

Figure 2.10 shows the relevant concepts of System of Patterns:

- *SepmSystemOfPatterns.* is a set of individual pattern (abstract and concrete) with their relationships (References). Thus dependencies between specific problems can be considered in a comprehensive way.
- *SeReference.* This link is used to specify the relationship between patterns with regard to the domain and software life cycle stage in the form of a pattern language. For example, a pattern at a certain software life cycle stage uses another pattern

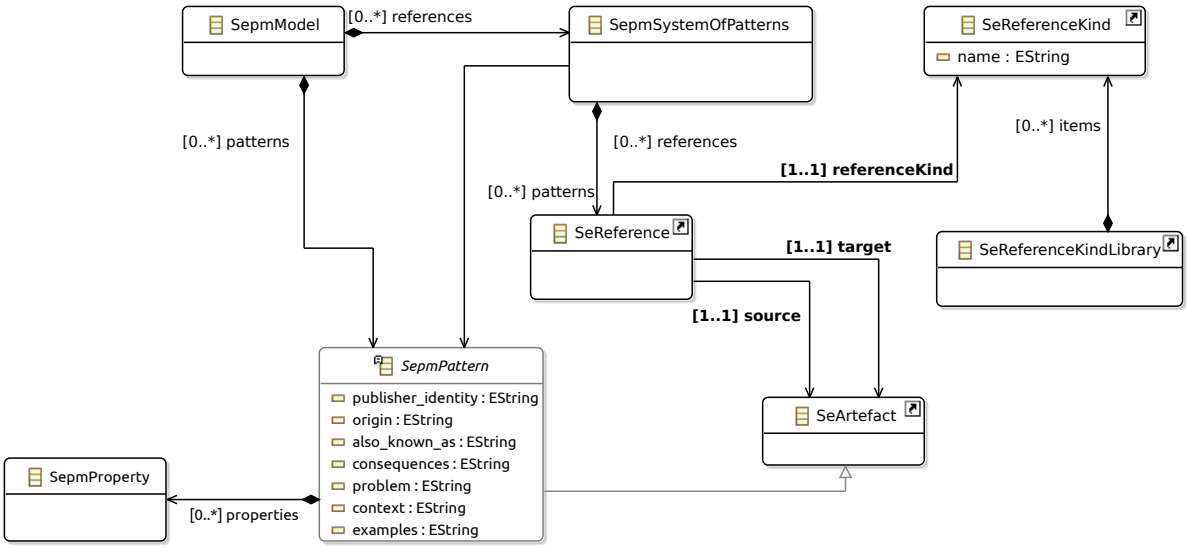


Figure 2.10: The (simplified) SEPM Metamodel: System of Patterns

at the same or at a different software life cycle stage. *SeReferenceKind* contains examples of these links. 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 [123].

- *refines.* It is used to represent the refinement relationship between two patterns.
- *specializes.* It is used to represent the specialization relationship (detail).
- *uses.* It is used to represent the functional dependency relationship between two patterns.
- *isSimilar.* It allows to link two patterns that perform the same functionality. This link is often used to link software patterns to their equivalent hardware patterns.
- *isAnAlternative.* It allows to link tow patterns that solve the same problem, but propose different solutions.

Example. We illustrate the usage of the SEPM for specifying a pattern with the example of secure communication pattern based on SSL[5] mechanism. Here, we specify an S&D property: "authenticity of sender and receiver". To type the category of this

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

property we use a category from the S&D category library: *Authenticity*. Moreover, we identify some resource properties, such as "CPU resource time for encryption" and "CPU resource time for authentication" that belong to category *CPUTime*, and "extra energy cost for encryption" and "extra energy cost for authentication" that belong to category *PowerConsumption*.

In our work, we focus on securing ICT software architectures during development using security patterns. Hence, we do not deal with *Pattern Development*. We assume that patterns are designed and validated by security engineering experts.

2.6 Introduction to the Case Studies

This section focuses on the introduction of the domain applications, which are addressed within the illustration and assessment of the proposed approach.

2.6.1 Working Example: Microsoft's web application

We use a web application example presented by Microsoft in [110] as a working example to illustrate the different contributions. Figure 2.11 shows the high level architecture of the web application. It consists of a client, a web server and a database server. The application includes assumptions on the trust boundary. It is a boundary where components can trust other components but cannot trust others outside it. In this case, it was considered that the web and data servers are in same trust boundary.

2.6.2 SCADA system

This case study is used for the assessment in section 7. SCADA systems are meant to control processes through local controllers, acquiring field data and returning them to a SCADA master computer system. Figure 2.12 shows a typical SCADA system architecture. It consists of a SCADA master, an operator workstation and a number of field devices connected by a communication infra-structure. Field devices can be Programmable Logic Controllers (PLC), Remote Terminal units (RTU), sensors and actuators. The SCADA master provides the operator with a Human-Machine Interface (HMI) through a work station to issue commands to PLCs and gather field data from them. PLCs are digital computers programmed to continuously monitor sensors and control actuators (e.g., valves, pumps, etc.). RTUs are used for converting sensor data into digital data. As SCADA systems cover large areas, they use Wide Area Networks (WAN). SCADA sys-

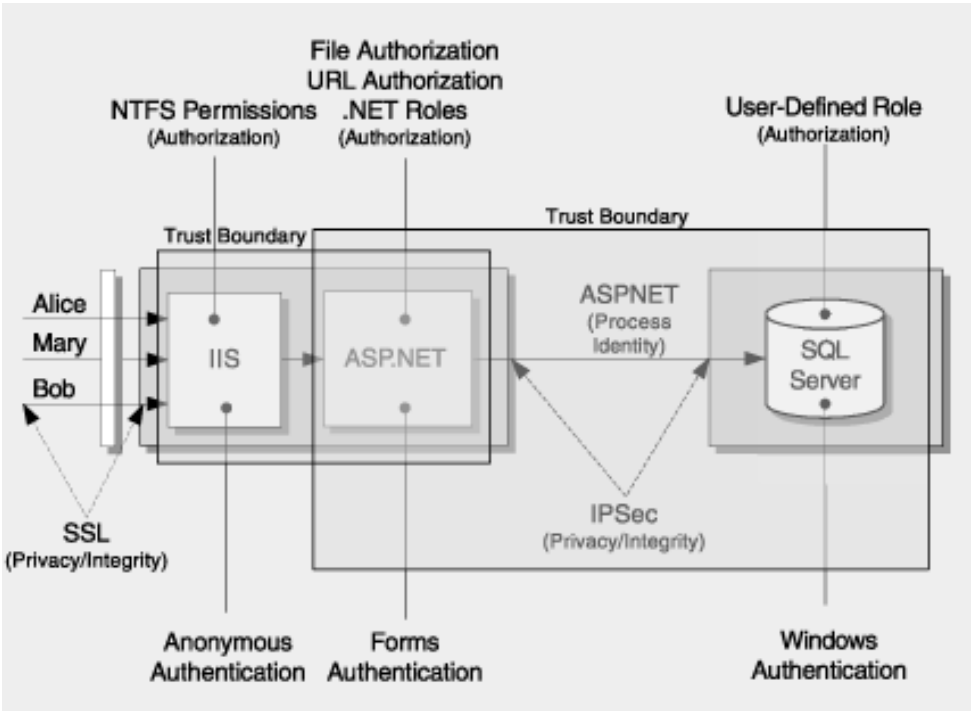


Figure 2.11: Middle tier web application architecture model [110]

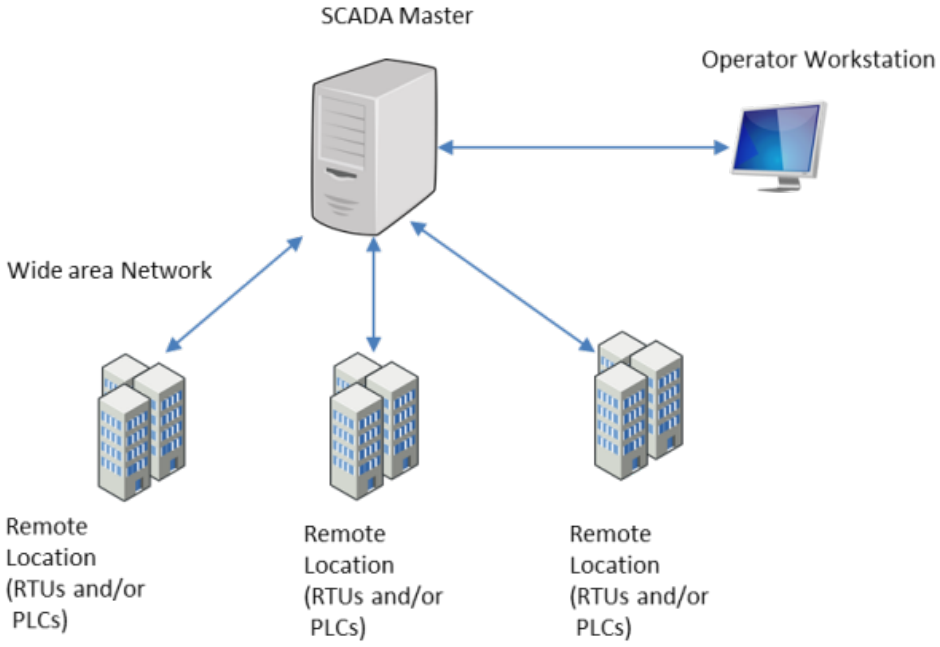


Figure 2.12: A typical SCADA system architecture [151]

tems provide the following features: data acquisition and handling (e.g., polling data from controllers, alarm handling, calculations, logging and archiving) on a set of parameters, typically those they are connected to.

2.7 Conclusion

MDE promotes models as a first class element. A model can be represented at different levels of abstraction and the MDE vision is based on (1) metamodeling techniques to describe these models and (2) mechanisms to specify the relations between them. Model exchange is within the heart of the MDE approach as well as the transformation/refinement relation between models. MDE frameworks may help software engineering specialists in their tasks, but indeed it would be interesting to use (partial) solutions and to guide them fulfilling recurring requirements. In software engineering, patterns meet this need. We leverage on this idea to propose a Pattern-Based Development process where architect can select, instantiate, integrate patterns and analyze software architecture solutions.

PBSE addresses challenges similar to those studied in software engineering focusing on patterns and from this viewpoint addresses two kind of processes: the process of *pattern development* and *system development with patterns*. In this work we focus on *system development with patterns* where the architect uses security patterns to secure ICT applications.

In this thesis, we propose to design and evaluate secure ICT applications using security patterns. The integration of security features requires the availability of both application domain specific knowledge and security expertise at the same time. Currently, the integration of security mechanisms is still new in many domains, hence embedded ICT architects usually have limited security expertise. We propose an integrated, MDE based tool supported approach for providing this expertise by the mean of security patterns. The proposed approach is based on selection, instantiation, integration of security patterns into the target application and a set of analyzes.

Chapter 3

Approach

Contents

3.1 Introduction

In this chapter, we give on overview of the approach coupled with a software development process. In this work, we illustrate the approach with a component-based development (CBD) process described in Figure 2.1. The remainder of this chapter is organized as follows. Section 3.2 describes the process of the approach mapped with a software development process. In section 3.3, we present some definitions essential for a common understanding of concepts. Finally, in section 4.7, we sum up and locate our contributions with regards of the activities of the approach.

3.2 Approach

The conceptual vision of our process model is visualized in Figure 3.1. In this vision, the software development process (Phases A1 to A4) is augmented with a risk assessment, risk treatment with patterns and real-time analyzes. The usage of this process proceeds as follow:

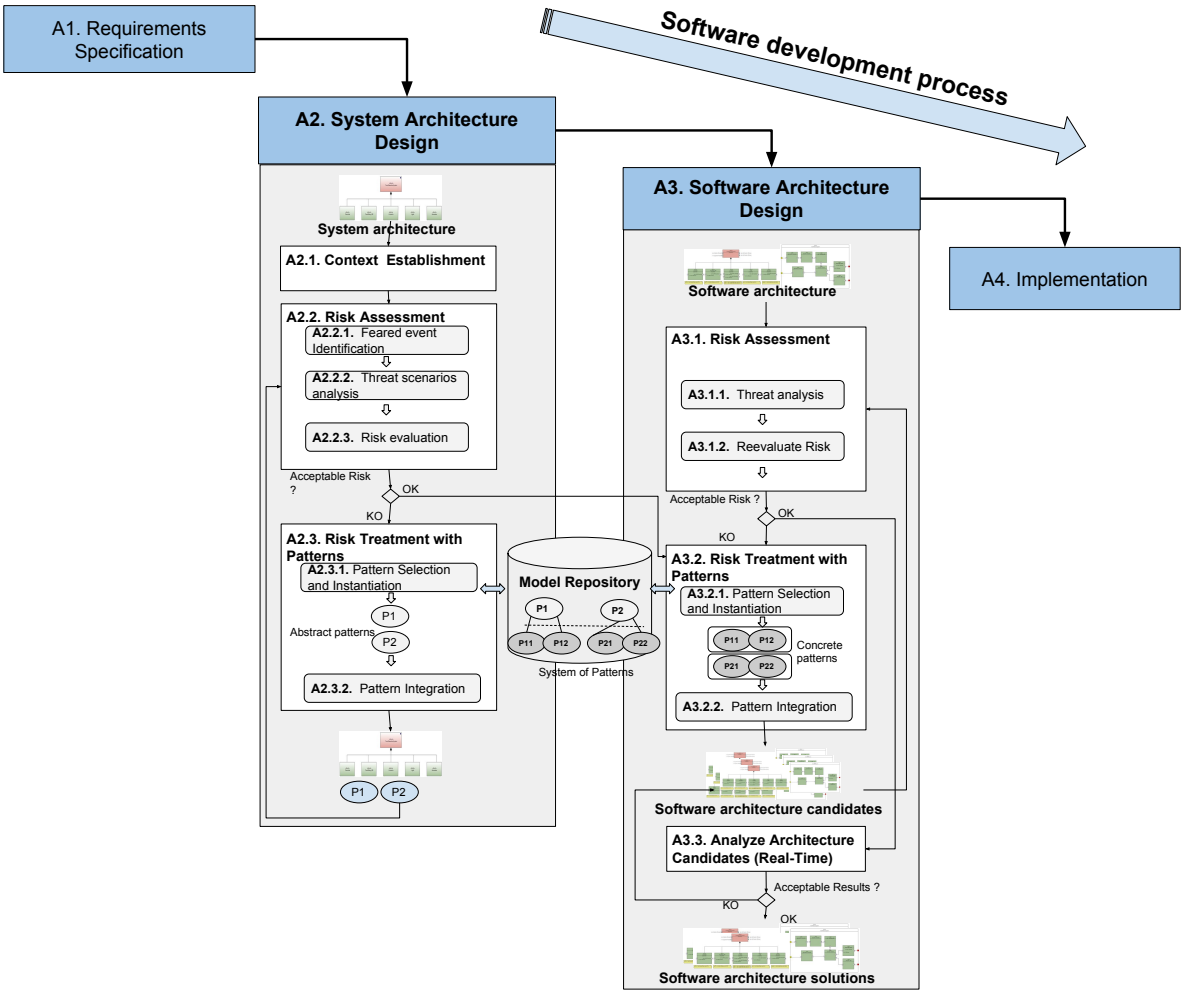


Figure 3.1: Approach mapped with Software Development Process focusing on System and Software Architecture phases

- Requirement Specifications (**A1.**). At this phase, requirement specifications are established (out of the scope of this thesis).
- System Architecture Design (**A2.**). The system architecture model is designed after performing functional analysis. The model is then submitted to risk management steps (EBIOS):
  - Context Establishment (**A2.1.**). The context of the analysis is set.
  - Risk Assessment (**A2.2.**). In this step, risk assessment is performed to enumerate and evaluate feared events, threat scenarios and risks at the system architecture. This process is done by a security risk analyst. Following this

step is the risk treatment phase.

- Risk Treatment with patterns (**A2.3.**). In this step, the architect interacts with the model repository based on risk assessment recommendations:
  - Pattern Selection (**A2.3.1.**). The selection of a System of Patterns relies on a model repository [61]. A System of Patterns consists of a set of security patterns (abstract and concrete) stopping identified threats. At the system architecture design phase, abstract patterns are instantiated in the modeling environment.
  - Pattern integration (**A2.3.2**). At the system architecture design phase, the integration of abstract patterns means specifying where the patterns are used. The process iterates from step (**A2.2.**) until risks are acceptable. Once risks are acceptable, the process moves to step (**A3.2.**).

- Software Architecture Design (**A3.**). At this phase the software architecture is designed by refining the system architecture.
  - Risk Assessment (**A3.1.**). The software architecture is submitted to risk assessment. Risk assessment at the software architecture level complements the one at the system architecture because software threats appear due to the level of details of the software architecture model. Hence, we formalize threat scenarios based on *STRIDE* [144] in order to detect threats in the software architecture. Concretely, a report is generated giving the number of software threats per threat scenario. Since new threats may appear due to level of details at the software architecture design level, identified risks may also be reevaluated.
  - Risk Treatment (**A3.2**) The risk treatment phase is done with:
    - Pattern Selection (**A3.2.1**). The selection of a System of Patterns consisting of a set of security patterns (abstract and concrete). At the software architecture phase, concrete patterns are instantiated in the modeling environment. These concrete patterns refine abstract patterns identified in the system architecture design phase. Since, different concrete patterns may refine a same abstract pattern, we obtain different System of Patterns configurations. Each System of Patterns configuration is a possible solution.
    - Pattern Integration (**A3.2.2**) At the software architecture phase, pattern integration consists of applying System of Patterns configurations on the

software architecture. Hence, since there may be different System of Patterns configurations, different software architecture solution candidates may be produced. The process iterates from step (**A3.1.**) until risks are acceptable.

– Real-time Analysis of Architecture Candidates (**A3.3**). The architecture solution candidates that pass activity (**A3.1.**), i.e. the risks are acceptable, are analyzed with regards to the impact of security patterns on the real-time aspects. In this case, scheduling analyzes are performed to verify that the software tasks respect their deadlines.

- Implementation (**A4**). This phase deals with the implementation of software components (out of the scope of this thesis).

Our scientific contributions are located in activities A2.3 and A3.2 to answer a part of: **RG1**, **RG2** and **RG4**, and A3.1 and A3.3 to answer a part of: **RG1**, **RG3** and **RG4**. The activities (A2.1 and A2.2) are essential to our work in the sense that their results are used as an input. However a complete tool chain support was developed to link all the activities (in the scope of the thesis).

3.3 Definitions

To allow common understanding in the context of the aforementioned approach, we have elaborated a set of definitions:

Definition 24 (Component). *A component is a modeling artifact which represents a piece of the software architecture. It has a set of required and provided interfaces.*

Definition 25 (System architecture). *A system architecture is a set of interconnected high level components.*

Definition 26 (Software architecture). *A software architecture is a set of interconnected software components refining the system architecture.*

Definition 27 (Application model). *An application model is the model of the system at two levels: system and software architecture levels (i.e., relevant to System Architecture and Software Architecture models respectively).*

In our context we refine **Definition 5** of a vulnerability :

Definition 28 (Vulnerability) *A vulnerability is a flaw in the architecture that constitute a weakness.*

- *System vulnerability*: is a weakness of the *system architecture* due to the non-existence of a certain countermeasure.
- *Software vulnerability*: is a weakness of the *system architecture* due to the non-existence of a certain countermeasure or the incorrect integration of the countermeasure in the software architecture.

In our context we refine **Definition 6** of a Threat :

Definition 29 (Threat) *A threat is a potential attack by a threat agent using an attack method.*

- *System threat*: exploits a *System vulnerability*
- *Software threat*: exploits a *Software vulnerability*

3.4 Conclusion

In this chapter, we have presented a global view of the proposed approach. It associates a generic software development life cycle, Pattern-Based System Engineering (PBSE) and a set of analyzes (threat and real-time analysis). In addition, we have introduced a set of definitions necessary for the understanding of the next chapters. Next, we present our contributions relevant to activities A2.3 and A3.2 to answer a part of: **RG1**, **RG2** and **RG4**, and A3.1 and A3.3 to answer a part of: **RG1**, **RG3** and **RG4**. In addition we study the feasibility of these contributions to answer **RG5**.

Chapter 4

Risk Treatment with Patterns: Selection, Instantiation and Integration

Contents

4.1 Introduction

System and Software Architects usually have basic knowledge in security engineering but lack expertise and best practices to apply the correct recommendations issued by security risk assessment (if any). Hence capturing and providing this expertise by the way of security patterns has become an area of research in the last years. Security can be captured within patterns that provide reusable generic solutions for recurring security problems, here dealing with architectural problems. Recently a complete catalog of security patterns has been introduced by Fernandez [48]. Unfortunately there are two major issues. First, traditional security patterns are usually described as informal guidelines to solve a

certain problem using templates such as POSA [29] and GoF[52]. Hence even if security patterns have advantages in fostering reuse, using them is difficult in reality. In fact, There is a major gap of understanding between threats issued by risk assessment and the description of a security patterns to protect against these threats. Furthermore, another difficulty is added during their integration i.e., incorporation into the architecture. Due to manual security pattern integration, the problem of incorrect integration (one of the most important source of security problems) remains unsolved.

In this chapter, we tackle these problems. We present an approach and tool support for the selection, instantiation and integration of proper security patterns during architecture design based on risk assessment recommendations (to answer a part of: **RG1**, **RG2** and **RG4**), as visualized in activities (A2.2), (A2.3) and (A3.2) of Figure 3.1. Activity (A2.2) is supported with EBIOS method [14]. Its goal is to assess risks, feared events and system threats. The identified threats are then used as input to the next activities. Activities (A2.3) and (A3.2) are devoted to risk treatment with security patterns. Activity (A2.3) consists on the selection and instantiation of security patterns and their integration producing architecture candidates. The application modeling is based on accepted OMG standards (UML, its Profile extension mechanism and OCL).

The remainder of the chapter is organized as follows. In section 4.2, we present related work and discuss the positioning of our contributions. In section 4.3, we give an overview of the selection and instantiation activity based on the identified system threats. This step outputs a System of Patterns. In section 4.4, we describe the manner of integrating a security pattern into an application. In section 4.5, we describe the model-based framework and illustrate it with Microsoft's web application. In section 4.7, we conclude and sum up the contributions.

4.2 Related Work

Over the years there has been a noticeable divorce between pattern experts and pattern users [102]. On one hand pattern experts create and document patterns and on the other hand pattern users are rarely aware of relevant patterns. In addition, the latter does not have a good understanding about how to leverage and apply a pattern. Research relevant to pattern specification, selection and integration is discussed bellow.

4.2.1 Pattern Specification

Most of works focus on guidelines which are pattern-specific, which means that they specify the essence of patterns, in other words what the pattern does (e.g., describing the frame of the pattern and its implementation) and how to interact with another pattern (describing how to refine a pattern from another one and how to combine two patterns). In this context, DPML [103] and the work of [20] were interested in pattern composition and formalization. In [51], the authors were interested in the process of composing several security patterns and their impact with regards to the composed security property. The pattern also gives the reasons for design decisions [58], which means that it includes descriptions of not only *what*, but also *why* [21]. In that sense, A pattern modeling language can be considered pattern-specific oriented because its goal is to support the design of a pattern.

4.2.2 Pattern Selection

Selecting appropriate patterns is an essential phase during the development of secure software and systems. So, many works are tackling this subject. We present here a selection of them regarding several aspects of this concern. The discussion is thus organized according to: secure development methodologies, general pattern selection techniques, and more specific methods devoted to the domain of security issues.

In [47], the authors presented a secure development methodology where the selection of security patterns is aided with a multidimensional classification of patterns according to: life cycle stage, concern, domain, architectural level and response type. In [155], the authors have surveyed and compared relevant methodologies for distributed systems. The surveyed methodologies use different ways for the selection of security patterns. Some use ad-hoc fashion, i.e. directly from security requirements. Others base the selection on a structured catalog of patterns, following guidelines in the overall approach or according to predefined schemas and conceptual frameworks.

Other works have presented methods for selecting patterns. They used different techniques: goal oriented formalization [160], text classification [73], classification based on properties and threats [28, 12], and ontologies [45].

These works have been applied more specifically for security patterns, in [160], the authors formalized security patterns using Goal-oriented Requirements Language (GRL) [13] in order to assist a designer with the selection of security patterns. In [72], the authors proposed an automatic security pattern selection method based on text classification and learning techniques for the classification of security patterns. The process is automated

for selecting security patterns from security requirements. The experiments showed that Naive Bayes is the most appropriate learning technique for selecting security patterns. In [45], the authors proposed a security pattern selection method based on ontological mappings at two different levels: (1) at design level: between requirements for design-based developer profiles and (2) at implementation level: between threat models, bugs and errors for implementation-based developer profiles.

In [12], the authors provided a classification of security patterns for each development life cycle stage. For instance, they proposed generic security properties for the design phase. In [28], the authors introduced classification schemes based on application domains, security aspects (e.g., confidentiality, integrity, availability, accountability, authentication and access control) and pattern recognition needs.

4.2.3 Pattern Composition and Application

Pattern integration [58], [52], [29] is a difficult task. Indeed, all the elements of the pattern must be integrated in an application without compromising the system integrity and quality while guaranteeing the new properties introduced by the pattern. In [58] the authors say that *applying a design pattern may be understood as transforming the system from one stage of development to the next.* One of the difficulties is that the pattern integration is a human-centered task [52] where the human has to understand the context and consequences related to a specific pattern – An important issue is given in [89]: *tools that work with patterns would have to be able to semantically understand your design as well as the pattern's trade-offs.* Furthermore, the integration topology must be taken into account. Eliel Saarinen (a Finnish-American architect and city planner) said: *Always design a thing by considering it in its next larger context - a chair in a room, a room in a house, a house in an environment, an environment in a city plan.* Once a pattern is integrated into an application, multiple problems can occur such as: side effects, functional regression, property needed is not reached, or yet patterns lose their essence because of multiple indirection cascade.

Works relevant to the integration of patterns and aspects are discussed since this issue (i.e., integration) has been tackled in both research areas.

Pattern integration

Buschmann [29], has defined three types of pattern collection: pattern catalog, pattern language and System of Patterns. The solutions proposed are based on kinds of patterns to describe pattern integration processes. In view of all this, Martin Flower [49] proposes

several types of integration styles. In SERENITY [4], integration schemes are defined to describe the pattern integration – *An Integration Scheme is an S&D Pattern that describes a complex S&D Solution.* The implementation of these solutions can be realized by following guidelines and according to Buschmann [29], with pattern collections such as pattern catalog, pattern language and System of Patterns.

In [143], the authors explained how pattern integration can be achieved by using a library of precisely described and formally verified solutions. In [136], the authors present an approach for creating of a security-enhanced system model using the SecFutur Engineering Process and the SecFutur Process Tool (SPT). In [101], the authors introduced a method and tool support called for developing secure and private IT systems using COmputer Supported Security Patterns (COSSP). The integration process targets object oriented applications. The tool support allows merging the solution of the pattern and the application and validating the result against a set of OCL rules in order to guarantee correct integration.

Aspect integration

Aspect-Oriented Modeling is similar to pattern modeling with regards to encapsulating concerns such as security for use. However the difference is that aspects are part of a software fulfilling a function dealing with the design stage, whereas patterns can deal with different development stages.

Some works focused on modeling security patterns as reusable aspects and integrating them in the design application. These works use weaving techniques and some form of Verification & Validation to prove that the aspect has been correctly integrated.

In [122], Nguyen et al. presented a pattern-driven secure system development process combined with an aspect-oriented security design methodology. To use this approach, the designer is required to manually construct security solutions, as a set of security patterns, of the considered system and the definition of a mapping for integration purposes of these solutions into the model-under-development. The remaining activities, including the effective integration are generated automatically using model weaving. Mouheb et al. [120] developed a UML profile that allows modeling security mechanisms as UML annotated aspect models to be woven automatically into a UML design model. Horcas et al. [75] propose an Aspect-Oriented Modeling (AOM) approach to weave customized security models into an application using the Common Variability Language (CVL) and the Atlas Transformation Language (ATL).
These works have left dealing with the Verification & Validation activity for future work.

In addition, conflicts between the design and other architectural attributes, may occur during this task after the weaving (i.e., merge activity).

In [53], Georg et al. proposed an approach for modeling security mechanisms and attacks as aspects using UML. In order to prove that the integration is correct, they use model verification on the application composed with the attack model and the security mechanisms.
This work is similar to ours, the main difference is that the integration is at a lower level: at Object-Oriented Modeling Level. The Verification & Validation is similar as they use OCL to model the security properties. However, these properties must be created and specified for each application and for each used aspect. In our case the properties and constraints are modeled together with the security patterns. The properties and constraints become specific by using the bindings.

4.2.4 Positioning

Contributions related to selection and integration are positioned. First, the selection approach is based on works of pattern classification [47, 28] according to security properties and asset categories which is a subset of pattern dimensions. It adds the usage of risk assessment to derive security properties. Table 4.1 positions the integration approach with the aforementioned ones mainly the work of: Ngugyen et al. [122], Mouheb et al. [120], Horas et al. [75] and Georg et al. [53]. We list a set of criteria that an aspect/pattern integration approach must hold in order for the integration to be correct:

- Pattern properties and constraints are considered in the approach and specified using a formal language *(Criterion 1)*.
- Pattern preconditions and postconditions should be application-independent *(Criterion 2)*.
- The application should be validated against postconditions *(Criterion 3)*.

Approaches	*(Criterion 1)*	*(Criterion 2)*	*(Criterion 3)*
Our approach	✓	✓	✓
Nguyen et al. [122]	×	×	×
Mouheb et al.[120]	×	×	×
Horcas et al. [75]	×	×	×
Georg et al. [53]	✓	×	✓

Table 4.1: Positioning of our contribution with regards to pattern/aspect integration processes

4.3 Patterns Selection and Instantiation

In this section, We first present the model repository structure and then the access tool allowing to selection of security pattern models.

4.3.1 SEMCO Model repository

Model repository structure

The model repository in Figure 4.1 is populated with a set of modeling artifacts:

- *Security patterns* abstract (domain independent) and concrete (domain specific) providing a set of security properties.
- *Property models* resource or security property models
- *Threat category models*: are categorized according to defined libraries. In our context, our categorization is based on STRIDE categories. STRIDE [144] refers to the following categories: Spoofing, Tampering, Repudiation, Information Disclosure, Denial Of Service (DoS) and Elevation of privilege. We choose to focus on the following categories: Man-In-The-Middle (MITM), Tampering, Injection and Denial of Service (DoS). This is neither a comprehensive nor a complete list but we tried to cover well-known categories frequently used. For instance, the injection category includes SQL injections, OS command injections and XPath injections and is ranked number 1 in OWASP top 10 [10].

Pattern Modeling Artifacts

The structure of the pattern consists of the following modeling artifacts :

- **Security Pattern** represents a modular part of a system that encapsulates a solution of a recurrent security problem in a specific context. It can be either abstract or concrete (see Definition 16 in section 2.5.3).
- **Security Pattern Solution** represents the architectural solution of the pattern. It consists of a number of software components: *participants* and *security mechanisms*.
- **Participants (roles)** represent generic components of a *security pattern solution*. They are the roles potentially played by components of the application.
- **Security mechanisms** are software components part of *security pattern solution*. They provide primitive security functions (e.g., encryption, signing). We provide a library of these functions in order to allow security pattern solution modeling.
- **Security Property** is a property model relevant to security provided by the pattern in order to stop or mitigate one or more threats. Security property categories are [48]:
  - Confidentiality: is a property category related to the protection against unauthorized information disclosure.
  - Integrity: is a property category related to the protection against unauthorized modification of data.
  - Authenticity: is a property category related to guaranteeing the data coming from an expected party.
  - Availability: is the protection against Denial of Service (DoS).
  - Non-repudiation: is the protection against the ability of a user to deny their actions.
- **Constraints**. Assumptions which will have to be satisfied by the application.

Repository sources and Target Formalisms

Where does these artifacts come from ? The design of the artifacts comes from a set of sources:

- industrial applications domains [152]
- academic [48]

- standards [147][30]

What target formalism does it support ? It is intended for modeling design tools based on:

- Domain Specific Modeling Languages: UML profiles
- General-Purpose Languages: UML [130], SysML [125]

However for each target formalism, an access tool must be specified. In this context, an access tool for component-based architectures relying on a dedicated UML profile named *ComponentUML*. This profile is described in section 4.5.1.

4.3.2 Access Tool

Here we focus on the repository accessing techniques providing simple interfaces. The model repository is accessed through an access tool intended for *Papyrus*.

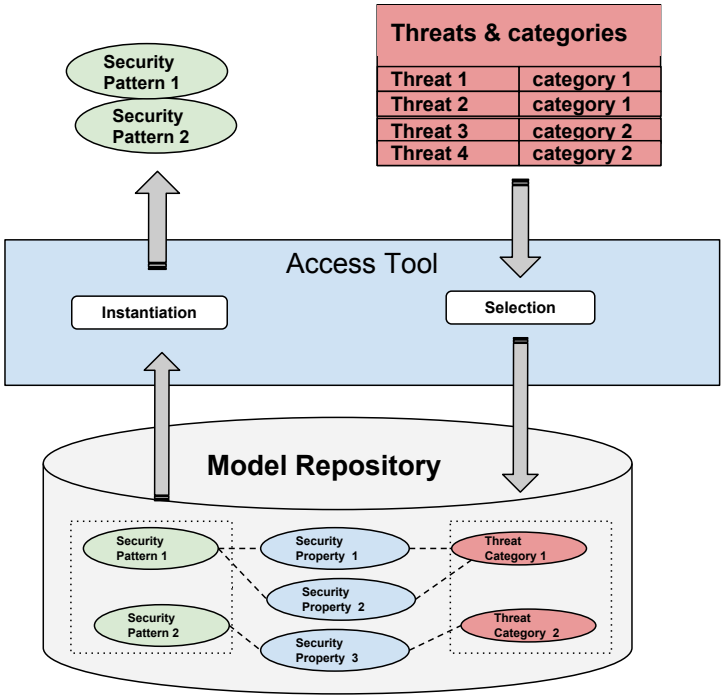


Figure 4.1: Selection and Instantiation of security pattern models according to threat models

The Access tool in Figure 4.1 allows selection and instantiation of patterns from the model repository:

- Search for security patterns using:
  - *Keyword search*
  - *Context search* according to:
    - Development life cycle stage (e.g., Architecture, Design, Implementation)
    - Domain (e.g., DI, DS)
  - *Threat category*: by selecting one or more threat category. The architect needs to search for architectural security patterns giving solution for identified security threats. For this to happen the architect needs to categorize the identified threats at the output of Activity **(A2.2.3.)** according to a threat category in 4.3.1.
- *Search Results*: The access tool suggests a System of Patterns which is a set of *abstract security patterns* protecting against the specified threats, their refinements *(concrete patterns)* and their relationships.
- *Artifacts Instantiation*: Instantiating patterns is a crucial step because it allows to translate the patterns from their formalism towards the UML-based formalism.

4.4 Pattern Integration

4.4.1 Methodology description

Once the System of Patterns is selected and instantiated in the UML-based modeling environment as mentioned in section 4.3, the fundamental question now is: Given a UML *security pattern* and an application constituted of a *UML application diagram*, *how to integrate correctly the first one into the second one ?* To respond to this question we see the security pattern solution as software architecture solution and we consider, as input of the process, patterns which are already validated. This process is based on previous work done in [66] which provides a first solution of design pattern integration in the context of object-oriented applications. We have further formalized the validation of security properties and constraints with OCL [126].

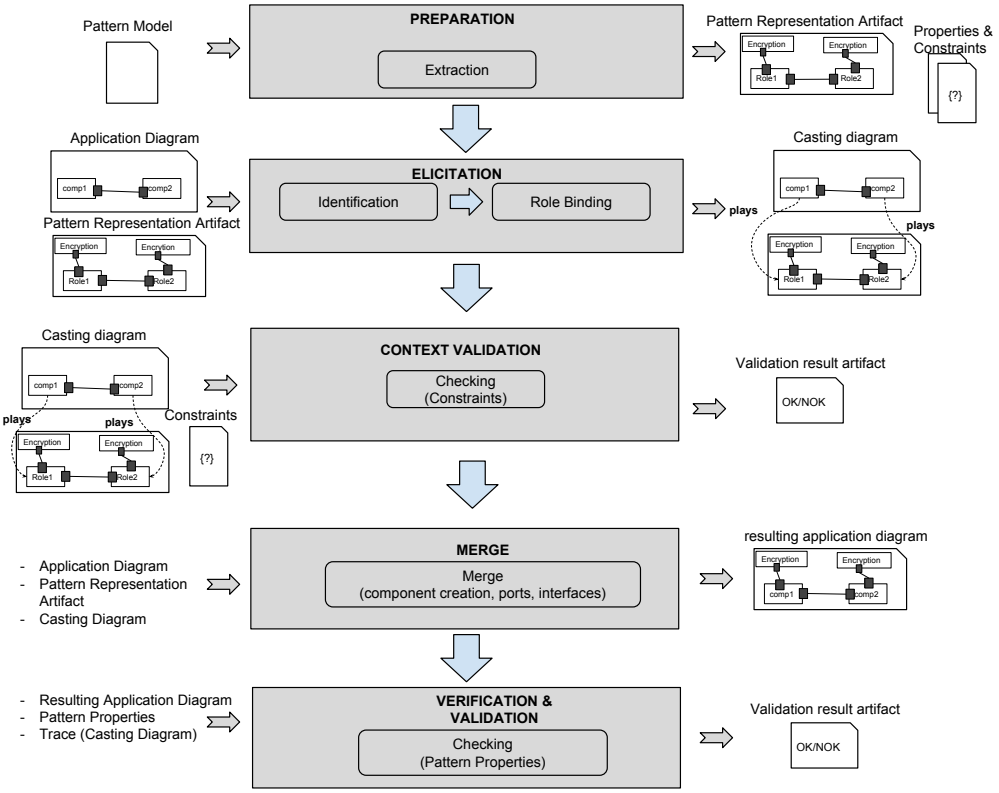


Figure 4.2: Pattern Integration Process

Figure 4.2 depicts our Pattern Integration Process which consists of five phases: Preparation, Elicitation, Context Validation, Merge and Verification & Validation. The phases are described after we present pattern integration artifacts consumed and produced by

each phase.

4.4.2 Pattern Integration Artifacts

The pattern integration process interacts with the following artifacts:

- **Application Diagram** is the representation of the software architecture of the application.
- **Pattern representation artifact** is the security pattern solution.
- **Preconditions** are the constraints that the application must verify in order for the integration to work.
- **Postconditions** are pattern properties that the application verifies after the integration of the pattern.
- **Pattern representation artifact** is the security pattern solution.
- **Casting Diagram** consists of the *application diagram*, the *pattern representation artifact* diagram and the *bindings* between components of the two diagrams. The bindings constitute mappings to identify the application components that play roles (participants) in the security pattern solution.
- **Resulting Application Diagram** is the final software architecture diagram once the pattern has been integrated into the application.

4.4.3 Hypothesis

We list a set of assumptions that we used to construct our integration process:

- *Hypothesis 1: Security feature incorporation.* The pattern integration process is only intended to incorporate security features. We verify that the application diagram contains the necessary functional features before starting the integration.

  For example: In order to integrate a secure channel Pattern between a Client and Server components:

  - There must be a communication.
  - Client and Server interfaces must have *Send()* and *Receive()*

- *Hypothesis 2: Pattern properties relevant to messages.* We focus particularly on the following pattern properties intended for transmitted messages:
  - **Property 1.** Confidentiality of messages
  - **Property 2.** Integrity of messages
  - **Property 3.** Authenticity of messages
- *Hypothesis 3: Pattern constraints.* In this context we focus on two major preconditions for all the patterns:
  - **Precondition 1.** All pattern participants should be bound (one participant per component)
  - **Precondition 2.** All communications in the pattern should exist in the application.

4.4.4 Phase 1: Preparation

The goal of this phase is to extract from a UML security pattern the necessary artifacts dedicated to the second phase: UML *pattern representation artifact, properties (postconditions), constraints (preconditions).* This phase can be considered as the warm-up of the pattern integration process during which the artifacts which are needed as real input of the process are prepared.

4.4.5 Phase 2: Elicitation

The aim of this phase is to identify where and how the pattern will be applied. Concretely, the Elicitation consists of: *identification* and *role binding*

- **Identification.** First the architect identifies the software components in the application diagram where the pattern representation artifact needs to be applied. This is done by searching for the components that shall play roles in the pattern representation artifacts.
- **Role bindings**. Once the software components are identified, the architect binds them with their corresponding *roles* in the patterns represented in Figure 4.2 by dashed arrows annotated by *plays* stereotypes.

4.4.6 Phase 3: Context Validation

The goal of this phase is to check whether the *pattern preconditions* are verified by the application. This phase takes as input the *Casting Diagram* and pattern *Preconditions* and provides as output a Result Artifact (that contains, for each constraint, the result of the checking). All constraints must be valid before moving to the next phase.

4.4.7 Phase 4: Merge

The aim of this phase is to correctly integrate the pattern using the *Casting Diagram*. The Merge phase consists of correctly adding *Security Mechanisms*, ports, interfaces and communications from the Pattern Representation Artifact into the application diagram. This results is a new version of the application. It is a candidate version for improved security of the application.

Let A be an application, *Pattern* a pattern representation artifact and C a casting diagram containing a set of bindings *bi*. The resulting application obtained by integration RA is defined by the algorithm in Listing 4.1.

```
1  Algorithm Merge
2
3  Input: A, Pattern, C.
4  Output: RA.
5
6  RA:= duplicate(A)
7  for each bi in C'
8    component1 = bi.component
9    patternParticipant1 = bi.patternParticipant
10
11   for each pPort in patternParticipant1
12    if pPort.communication.prts.component->includes(SecurityMechanism)
13           Prt1 = RA.duplicatePort(pPort,component1)
14           securityMechanism = RA.duplicateComponent(Port.securityMechanism)
15           RA.CreateCommunication(Prt1,securityMechanism.port)
16    else
17     patternPartipant2 = Port.communication.ports.component->select(PP | PP !=
           patternParticipant1)
18     component2 = patternParticipant.binding.component
19     Prt1 = RA.ports->select(prt| prt.owner = component1 and Prt1.communication.connects(
           component1,component2))
20
21     for each pOperation in Port.interface.operations
22      if Prt1.interface.operations->includes(pOperation) == false
23       RA.addOperation(pOperation, Prt1.interface)
24      endif
25     endfor
26
27    endif
```

```
28    endfor
29  endfor
```

Listing 4.1: Merge algorithm

Application diagram duplication (Line 6-11). First, the application diagram A and casting diagram C are duplicated and named RA and C' respectively. The set of bindings in C' are parsed. For each binding *bi*, *patternParticipant*1 and *component*1 are the pattern participant and the application component bound by *bi* respectively. In addition, ports owned by *patternParticipant*1 are looked up.

Security mechanism deployment (Line 12-15). These lines aim at deploying a security mechanism to application components.
For each port *pPort*, if *pPort* connects *patternParticipant*1 to a security mechanism, then it is duplicated, added to *component*1 in RA and named $Prt1$. The security mechanism is duplicated, added to RA and named *securityMechanism*. A communication is created between Prt1 and *securityMechanism.port* (*securityMechanism* port).

Security mechanism deployment (Line 16-23) These lines aim at adding the necessary operations to the interfaces of the application components in order to correctly call the operations of the security mechanisms. If *pPort* does not connect *patternParticipant*1 to a security mechanism, then it is connected to another pattern participant that we name *patternParticipant*2. We name *component*2 the application component bound to *patternParticipant*2. According to the assumptions, there must be a communication between application components (this is a precondition). In this case, *component*1 is connected to *component*2 via port $Prt1$. The operations of *pPort* interface are looked up. For each operation *pOperation* in *pPort* interface, if *pOperation* does not exist in $Prt1$ interface, *pOperation* is added to the operations of $Prt1$ interface.

After the merge phase, the resulting application verifies the *pattern postconditions*

4.4.8 Phase 5: Verification & Validation

This phase consists in checking the resulting application model against the *pattern postconditions* at each model change. The verification of postconditions (pattern security properties) over the new application is guaranteed by the checking module. Changes, after the integration, can result from ad-hoc tailoring or from the integration of another security pattern. The validation of the application against pattern postconditions is done by verifying that:

- The necessary security mechanisms ensuring pattern properties exist in the application. For example: the existence of the encryptor mechanism for confidentiality
- The security mechanisms are correctly used to protect messages. For example: the encryption of a message to ensure its confidentiality during transmission.

4.5 MDE Framework

In this section, we describe an MDE framework to support activities Risk Assessment and Pattern Selection and Integration. We use metamodeling, existing modeling languages and Model-To-Model transformation techniques for building a secure system and software architecture model. For the illustration, we consider the web application presented in section 2.6.1.

4.5.1 Architecture Design Modeling

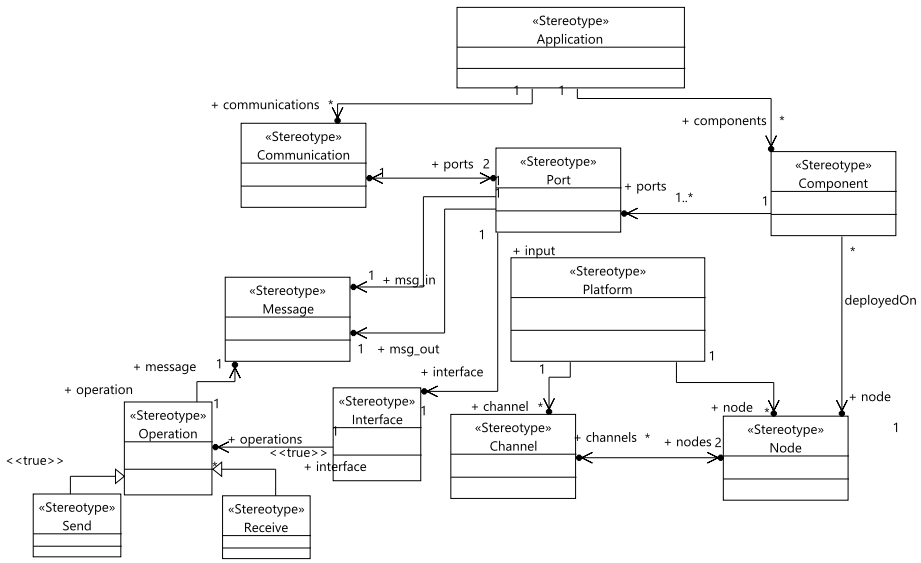


Figure 4.3: UML profile for component-based software architectures

In the context of the CBD process presented in section 2.2.2, we are interested in component-based architectures. We defined the UML profile in Figure 4.3 in order to model the application (i.e., system and software architecture). The need to define this

profile occurred during the formalization process of preconditions and postconditions, and later threat scenarios, using OCL. The OCL rules were difficult when using UML because concepts that were not relevant appeared. Hence this profile was used for a matter of simplification. This profile is not a final solution. This point is discussed later in future works.

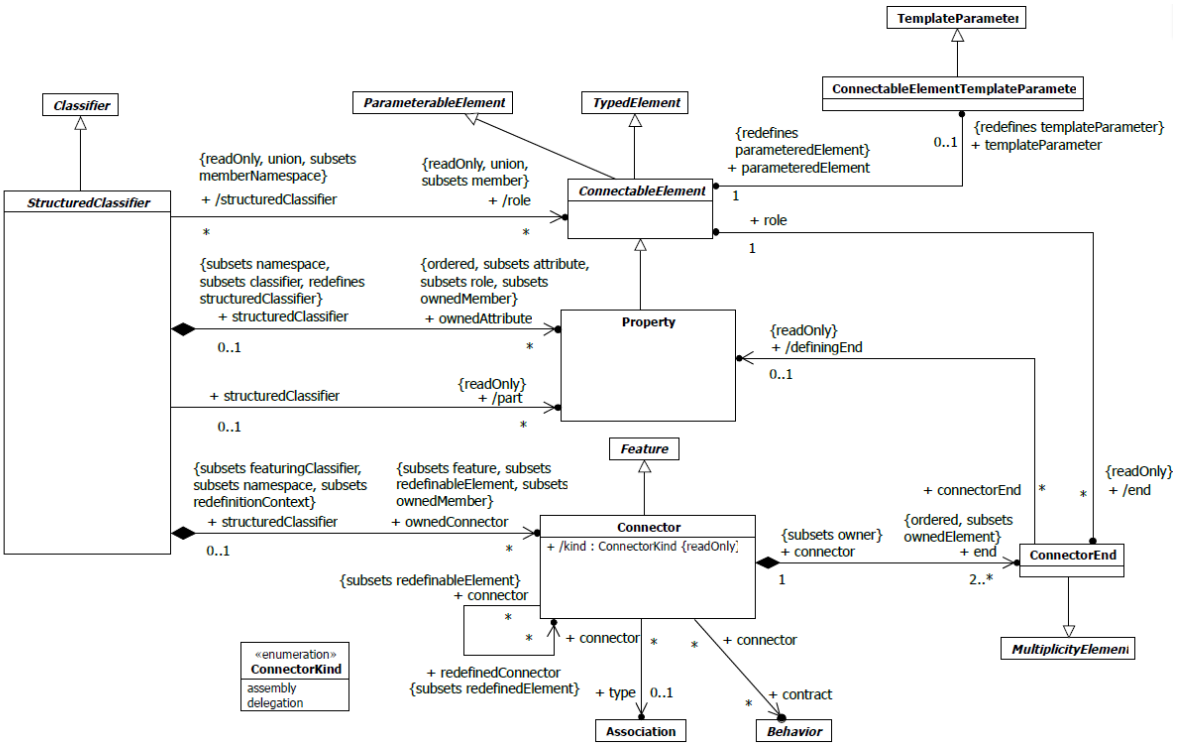


Figure 4.4: StructuredContainer from UML 2.5

The UML profile has been defined based on the following concepts: StructuredClassifiers (Figure 4.4), Messages (Figure 4.5) and Deployments from UML 2.5 [130] (Figure 4.5). When we study distributed systems, often we use models to denote some abstract representation (environment) of a distributed system. To encode distributed computing (protocols) in such systems, we use message passing model [17], where system components have only local vision of the system and communicate only with their neighbors by messages. The program executed at each node consists of a set of variables (state) and a finite set of actions. A component can write to its own variables, send and/or receive messages from its neighbors. The stereotypes and the UML extensions are given in Table 4.2:

The architecture design of the application is modeled at two levels as displayed in Figure 3.1 using ComponentUML:

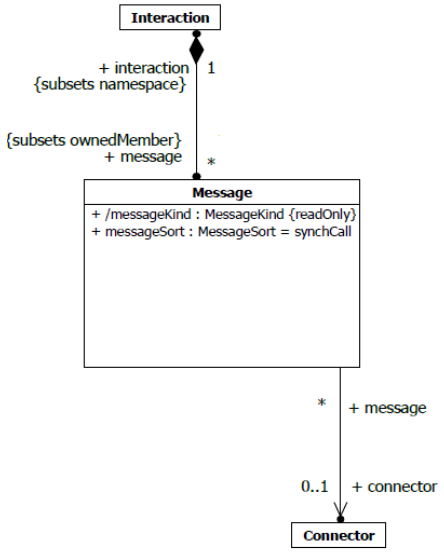


Figure 4.5: Messages from UML 2.5

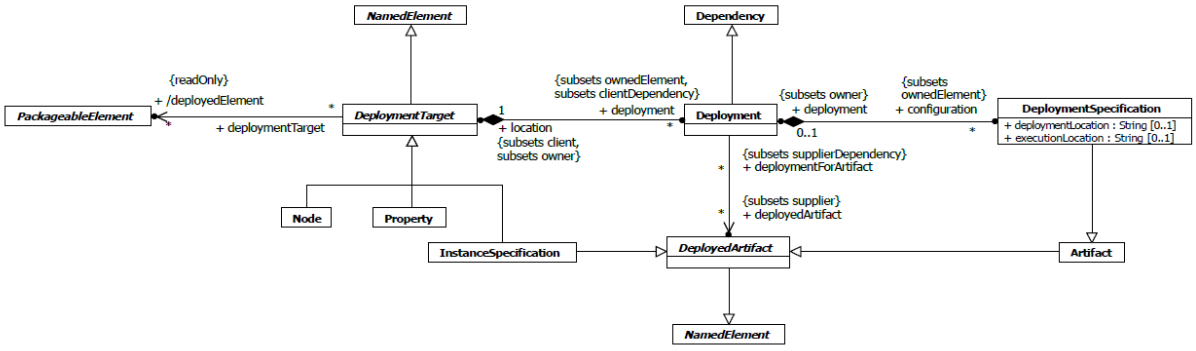


Figure 4.6: Deployment from UML 2.5

Stereotypes	UML extensions
Application	Class
Component	Property
Port	Port
Interface	Interface
Operation	Operation
Send	Operation
Receive	Operation
Communication	Connector
Message	Property
Platform	Class
Node	Property
Channel	Connector

Table 4.2: ComponentUML stereotypes and extensions

System Architecture Design. At this stage, the application model consists of system components (independent of the software and hardware choice) connected with communications.

Figure 4.7 shows the system architecture of the web application. It is composed of two system components: (1) client and (2) website.

Software Architecture Design. The software architecture model is obtained by refining the system architecture model. Hence, Ports, Interfaces , operations and messages are added to the application. The platform consists of a set of nodes connected with channels.

Figure 4.8 shows the software architecture of the application. Ports, interfaces, data types, and messages are added. The component website is refined into two software components: (1) webserver and (2) database.

4.5.2 EBIOS Risk Assessment

We have implemented a framework that supports the EBIOS method and integrated it into our modeling environment Papyrus. This framework includes a UML profile and tools to automate some steps in the EBIOS method (generation of an attack tree). We can also generate tables (list of risks, feared events and system threats) specified in the EBIOS method.

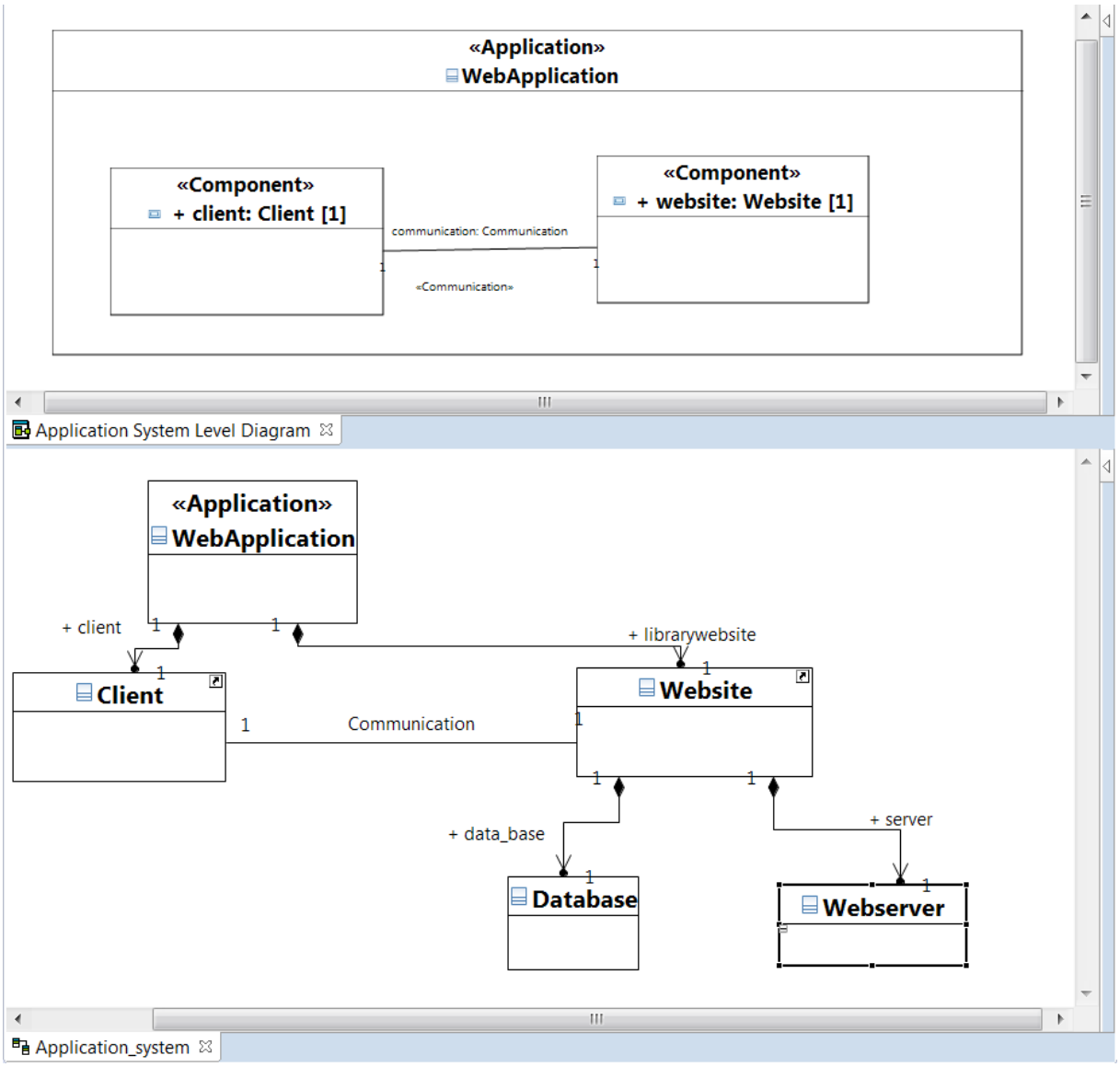


Figure 4.7: Web application system architecture model

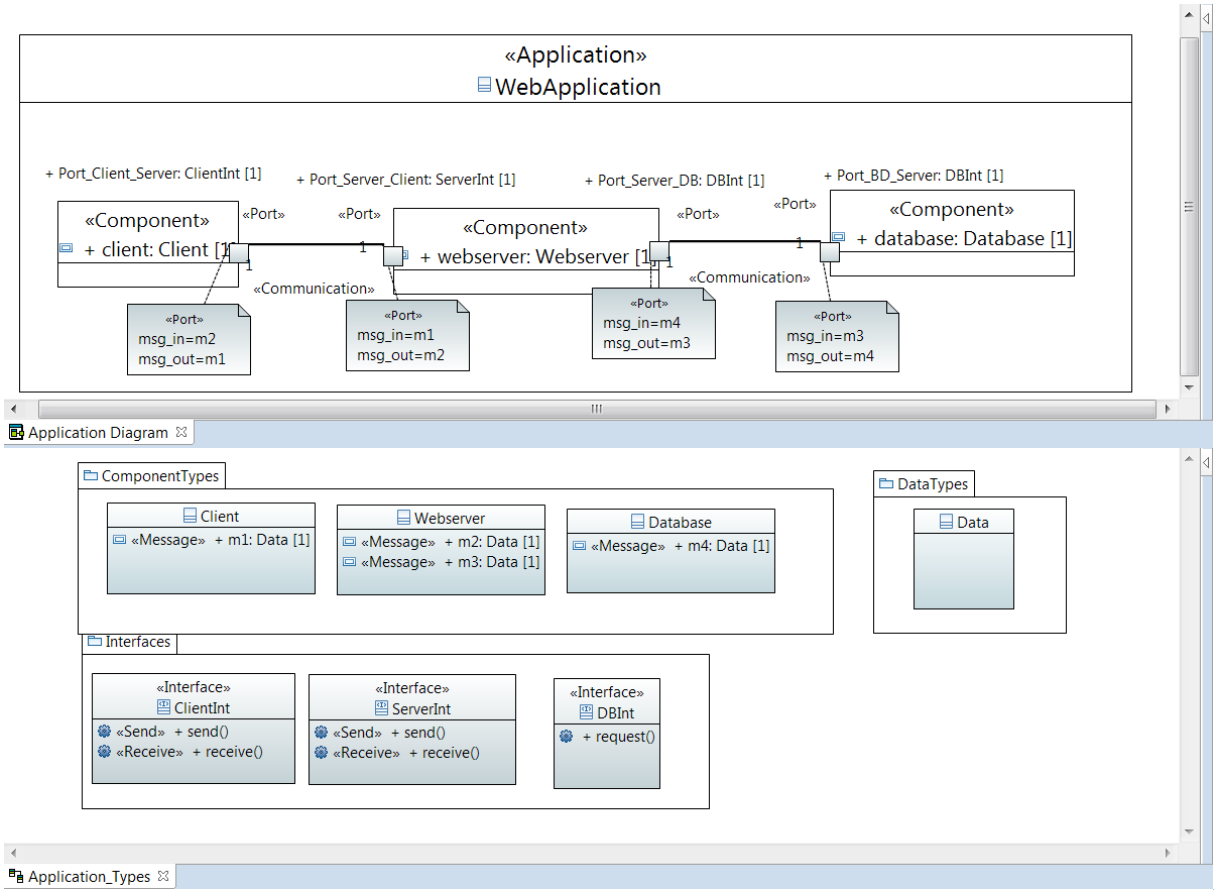


Figure 4.8: Web application software architecture model and Types

EBIOS UML profile

Figure 4.9 presents an excerpt of EBIOS UML profile. It contains all the concepts presented in section 2.3.2.

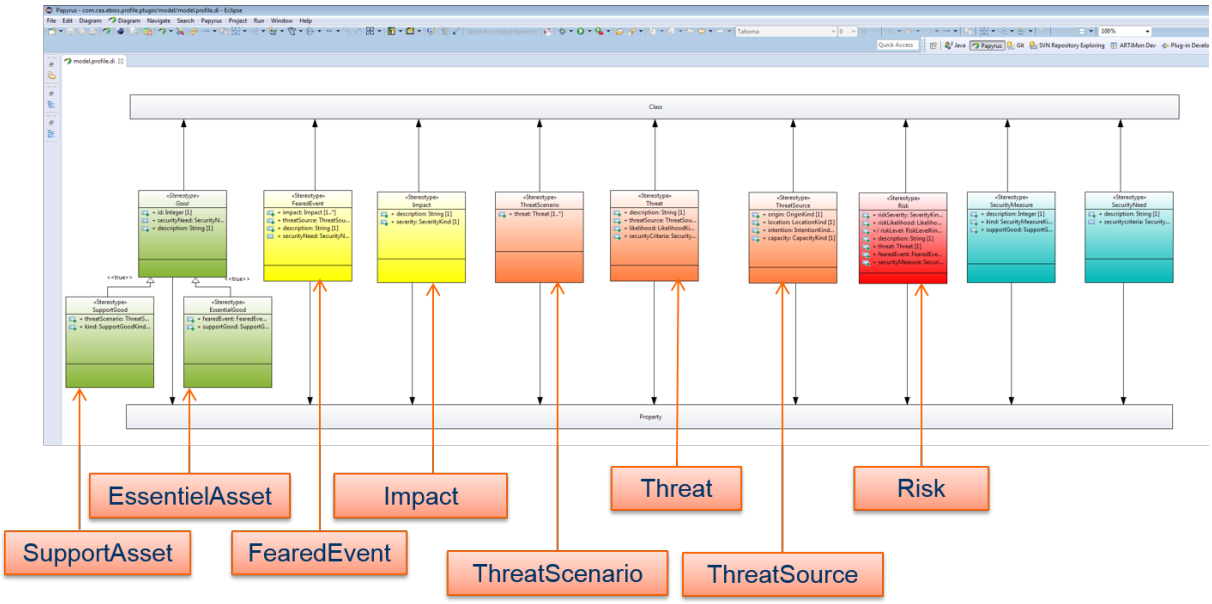


Figure 4.9: Excerpt of EBIOS UML profile

Illustration

Figure 4.10 shows the different diagrams used to model the system architecture (diagram **(1, 2)**), feared events and associated system threats (diagram **3**) and a table view of feared events and system threats (diagram **4**) with their likelihood and impact levels. Figure 4.11 shows an excerpt of automatically generated Attack Trees based on the results of the EBIOS risk assessment. Each attack tree describes a risk within the application. In this Attack Tree, the top nodes (Communication disruption, Client disruption, Website Disruption) correspond to a feared event in the EBIOS analysis. Following this node is an OR node. Children nodes corresponding to the system threats that may lead to the feared event.

4.5.3 Selection and Instantiation

Once the system threats are identified, the selected System of Patterns in the pattern repository needs to be instantiated into the modeling environment. We describe the target UML profile (SepmUML) and the mappings necessary for the instantiation.

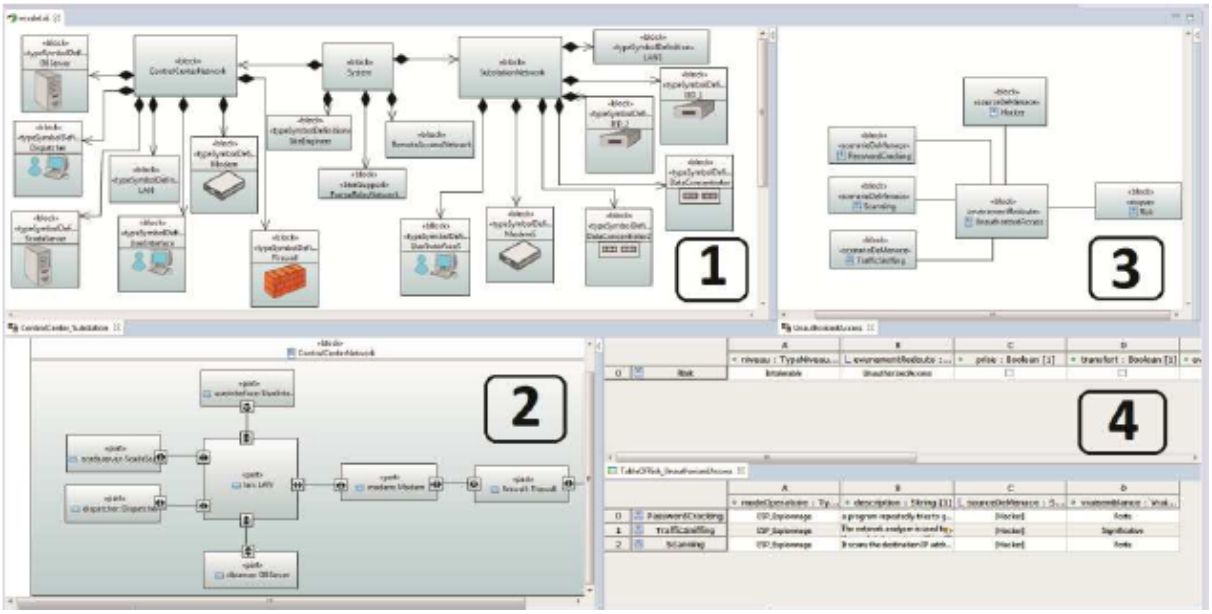


Figure 4.10: EBIOS analysis diagrams

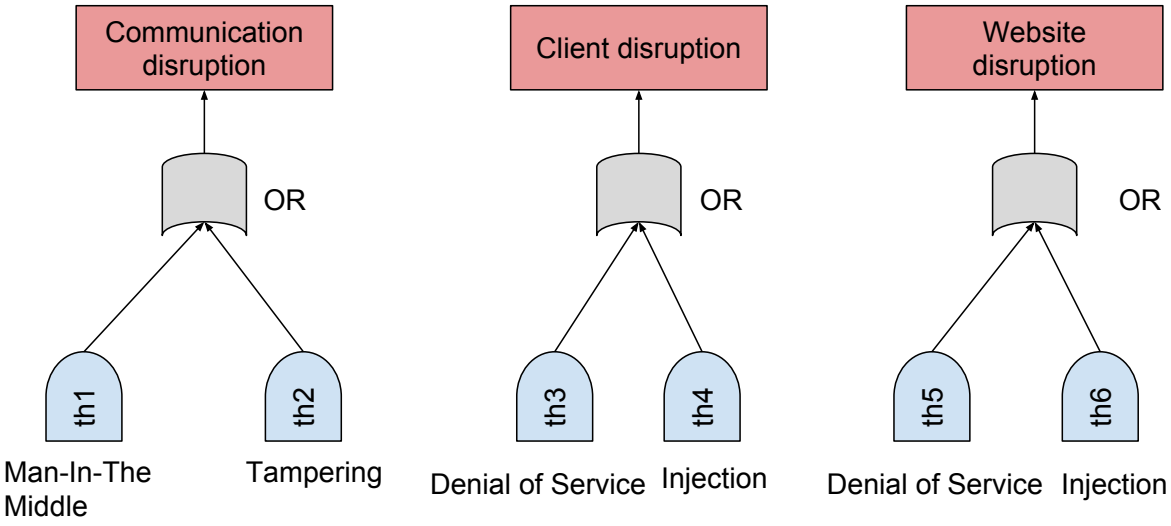


Figure 4.11: Excerpt of web application attack trees

SepmUML: UML profile

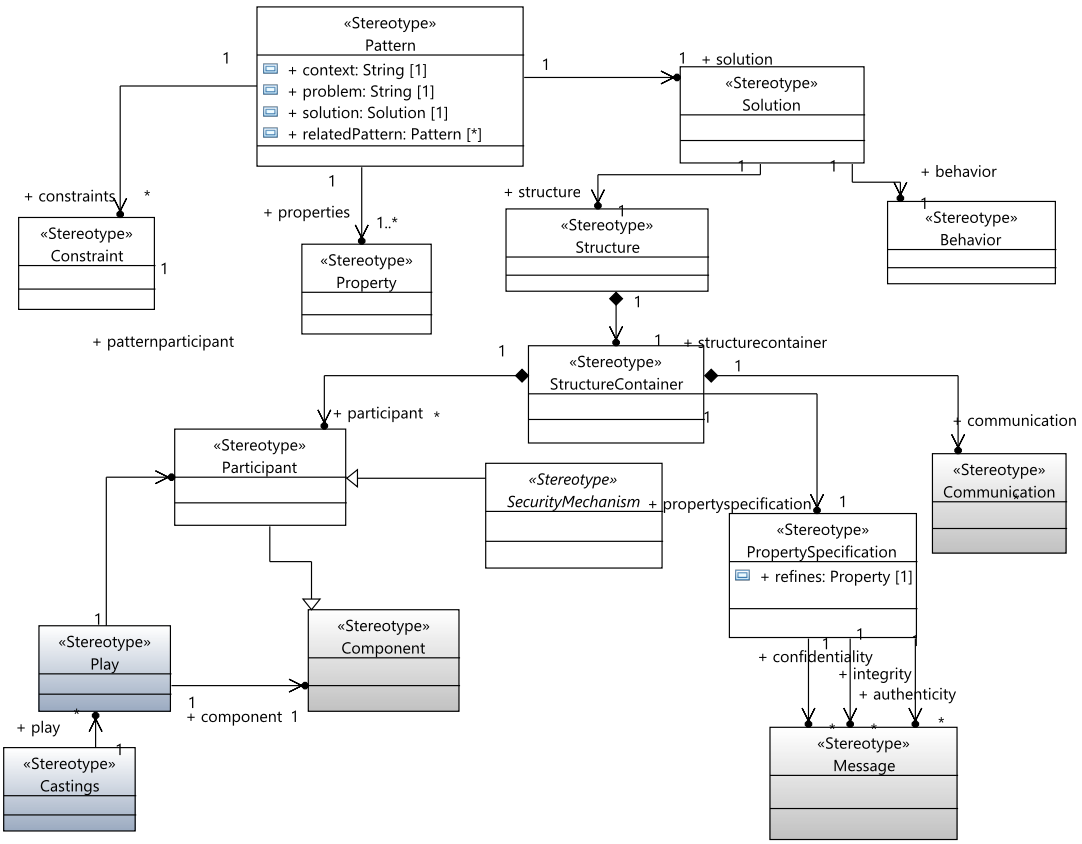


Figure 4.12: SepmUML UML profile

SepmUML is a UML profile for modeling security patterns in the context of CBD (see Figure 4.12). SepmUML contains the necessary stereotypes for modeling a security pattern in UML environment (stereotypes in white). It extends the SEPM conceptual model presented in section 2.5.3 with integration related concepts (stereotypes in blue). SEPM is a generic metamodel, hence the target application modeling language is not specified. The solution of the pattern is modeled using ComponentUML (stereotypes in Grey). Figure 4.13 shows the stereotypes relevant to a System of Patterns based on SEPM in Figure 2.10. Table 4.3 shows SepmUML stereotypes and UML extensions.

SepmUML Security Mechanisms Library

Five main security mechanism categories of SempUML are considered: Authenticity, Authorisation, Authentication, Cryptography, Monitoring and Filtering as it can be seen in

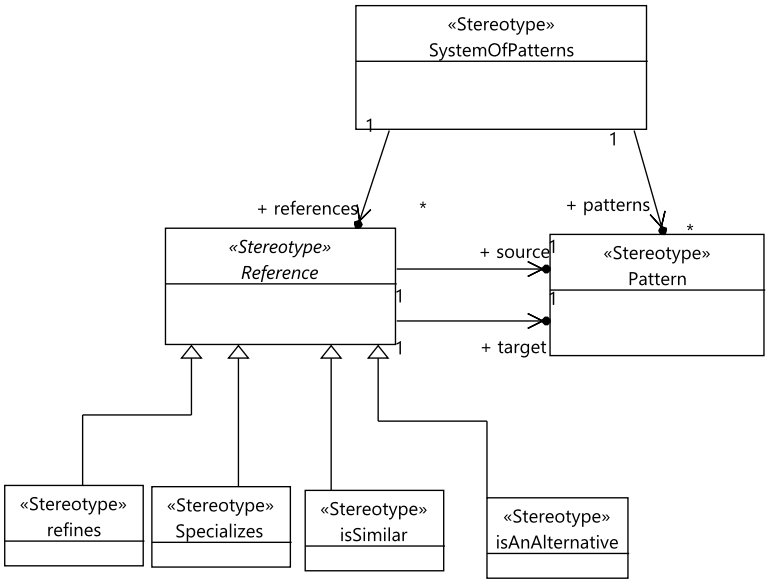


Figure 4.13: SepmUML UML profile: System of Patterns

Stereotypes	**UML extensions**
Pattern	Package
Property	Comment
PropertySpecification	Comment
Constraint	Comment
Solution	Package
Structure	Package
StructureContainer	Class
Behavior	Package
Participant	Property
SecurityMechanism	Property
Play	Dependency
Castings	Package
PatternIntegration	Package

Table 4.3: SepmUML stereotypes and extensions

Figure 4.14. Derived from security requirements, a customised security pattern solution can be built up from a combination of these security mechanisms categories.

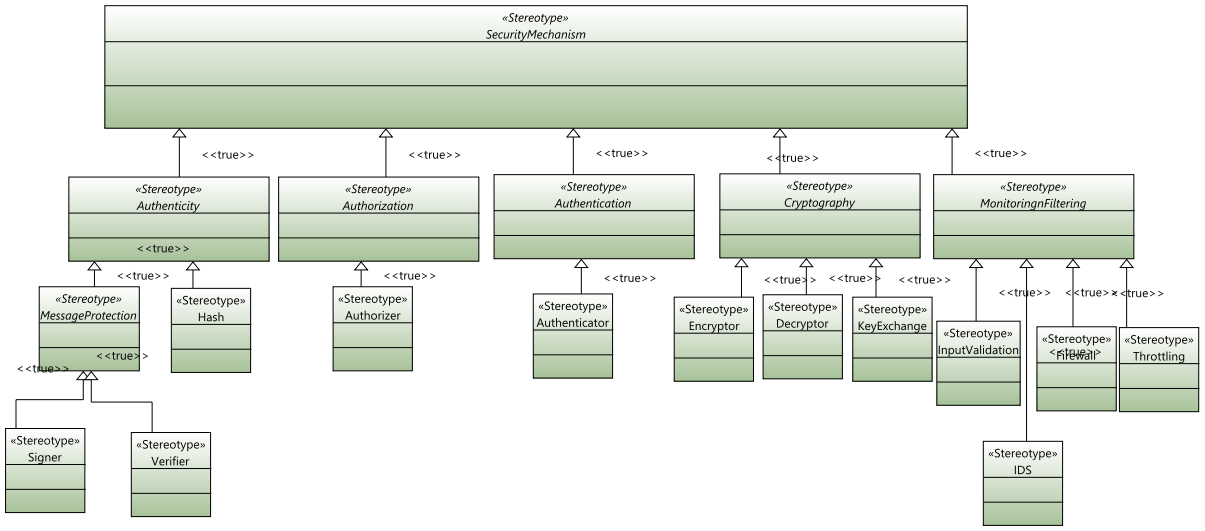


Figure 4.14: A partial view of the considered security mechanisms

Transformation for Instantiation: SEPM towards SepmUML+ComponentUML

System of Patterns need to be instantiated from the model-based repository SEMCO the UML modeling environment. We define mappings from SEPM to SempUML+ComponentUML (i.e., SempUML and ComponentUML) as shown in Table. 4.4 in order to instantiate one pattern. For instantiating a System of Patterns, the transformation is performed for each pattern and for each reference. These mappings are implemented using QVTo as shown in Figure 4.15. Figure 4.15 shows an overview of a set of transformation rules using QVTo under EMF. SEPM and SepmUML are specified using Ecore and UML Profiles respectively, as source and target metamodels for the transformation rules.

Source	Target
SEPM	*SepmUML/ComponentUML*
SepmPattern elements	Pattern
SepmProperty	Property
SepmConstraint	Constraint
SepmInternalStructure	Solution
SepmStaticInternalStructure	Structure/StructureContainer
SepmStaticInternalStructure	Behavior
SepmParticipant	Participant
SepmSecurityMechanism	SecurityMechanism
SepmInternalInterface	Ports/Interface
SepmOperation	Operation
SepmLink	Communication
SepmParameter	Message

Table 4.4: SEPM to SepmUML+ComponentUML Mappings

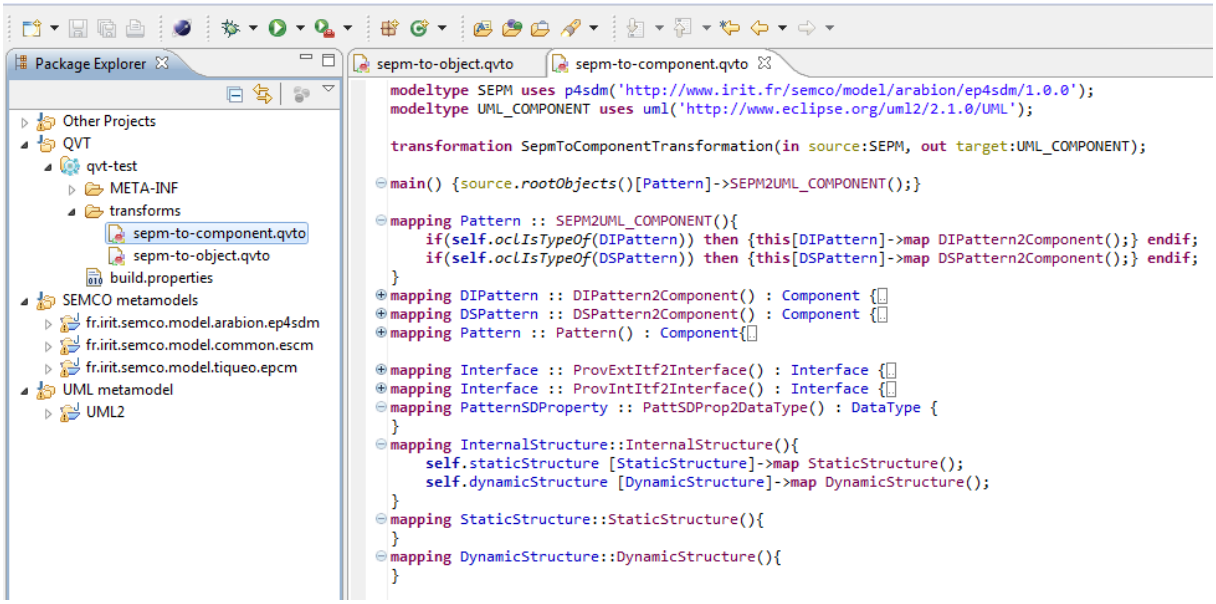


Figure 4.15: Mapping rules from SEPM concepts to SepmUML+ComponentUML using QVTo

Illustration

We illustrate the selection and instantiation activity for threats **th1** and **th2** shown in Figure 4.11. The objective is to protect the communication between the Website and Client against Man-In-The-Middle (MITM) (**th1**) and Tampering attacks (**th2**).

System of Patterns: Secure Communication (SSL, IPsec). The technical details of the access tool will be presented in section 4.6. The access tool suggests the System of Patterns as described in Figure 4.16:

- 1 abstract pattern: Secure Communication Pattern (SCP)
- 2 concrete patterns: SSL and IPsec patterns refining SCP

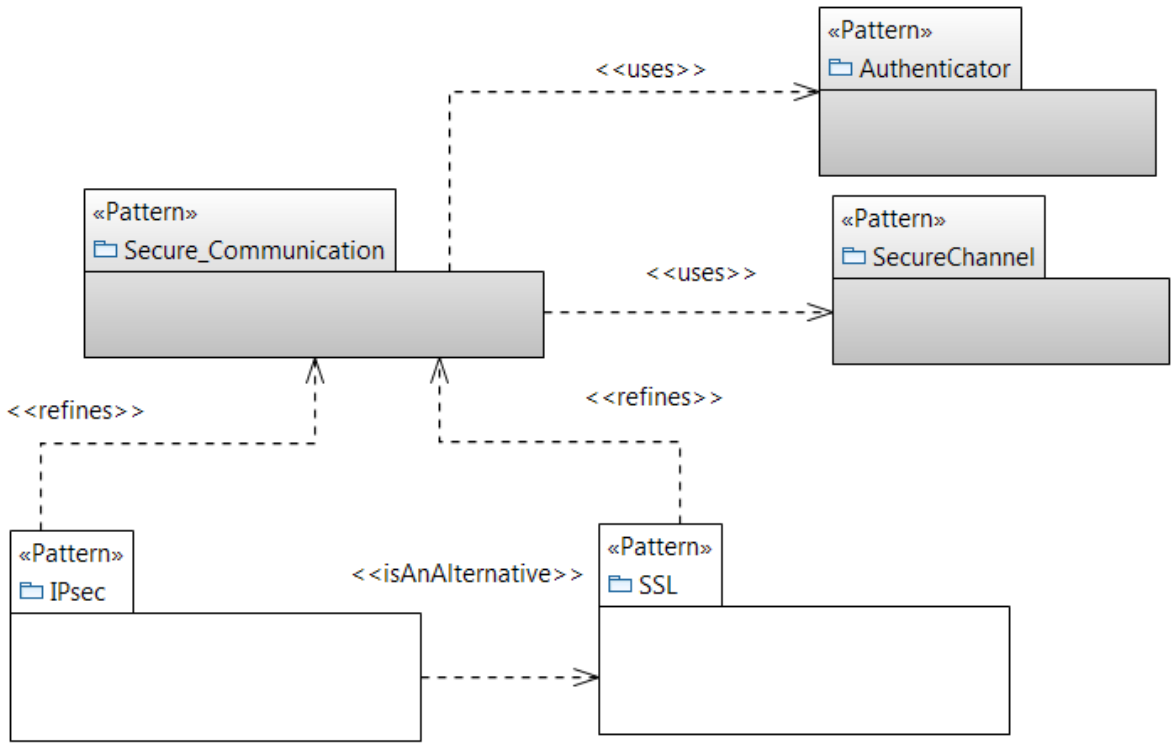


Figure 4.16: System of Patterns: Secure Communication (SSL, IPsec)

Using Figure 4.11, the abstract patterns are linked with the web application system architecture to identify where they are used. This is possible because each feared event is relevant to a system component and each threat is stopped or mitigated with an abstract pattern. Figure 4.17 shows where the abstract patterns are used with regards to the system architecture.

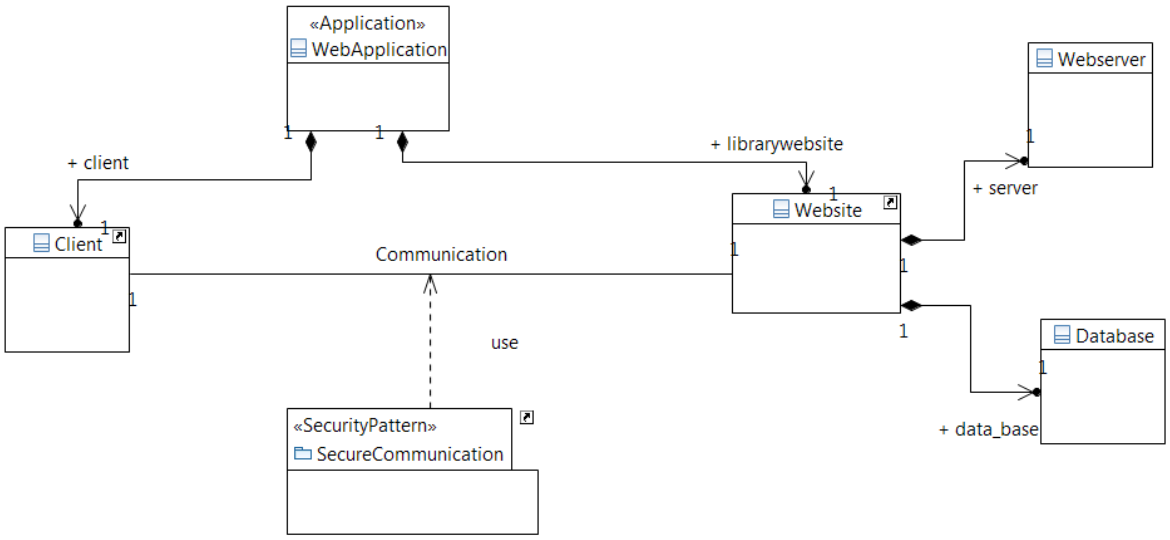


Figure 4.17: Web application system architecture with pattern usage

SCP has two refinements and thus two alternatives. Hence, there are two System of Patterns configurations (see Definition 19 in section 2.5.3). It means that after the next step (integration), we would have two produced architectures candidates. The goal of the SCP and its refinements is to ensure that the data is secure during transmission knowing that messages passing across any public network can be intercepted. These pattern are described in Appendix. A.

SSL Pattern model with SEPM. For illustration, purposes, we focus on the SSL pattern. Figure 4.18 shows the pattern model in SEPM.

SSL Pattern model with SepmUML+ComponentUML. The instantiation activates the Model-To-Model transformation in Figure 4.15 from SEPM into SepmUML+ComponentUML. The pattern contains: its *properties (Postconditions)* and *constraints (Preconditions)*. They are expressed with natural language to provide a quick understanding of the pattern. Figure 4.19 shows the pattern model in SepmUML.

4.5.4 Integration: Preparation

In this phase, the aim is the extraction of the pattern solution and pattern preconditions and postconditions.

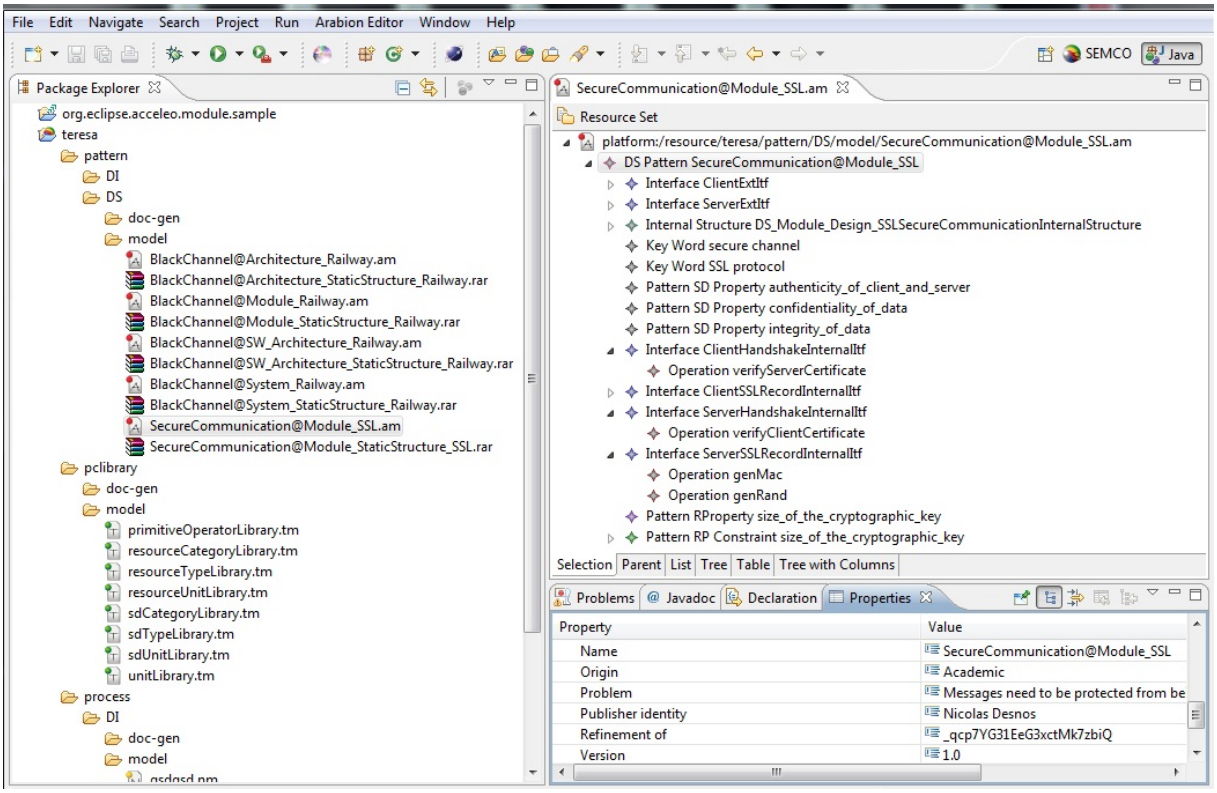


Figure 4.18: SSL Pattern: SEPM

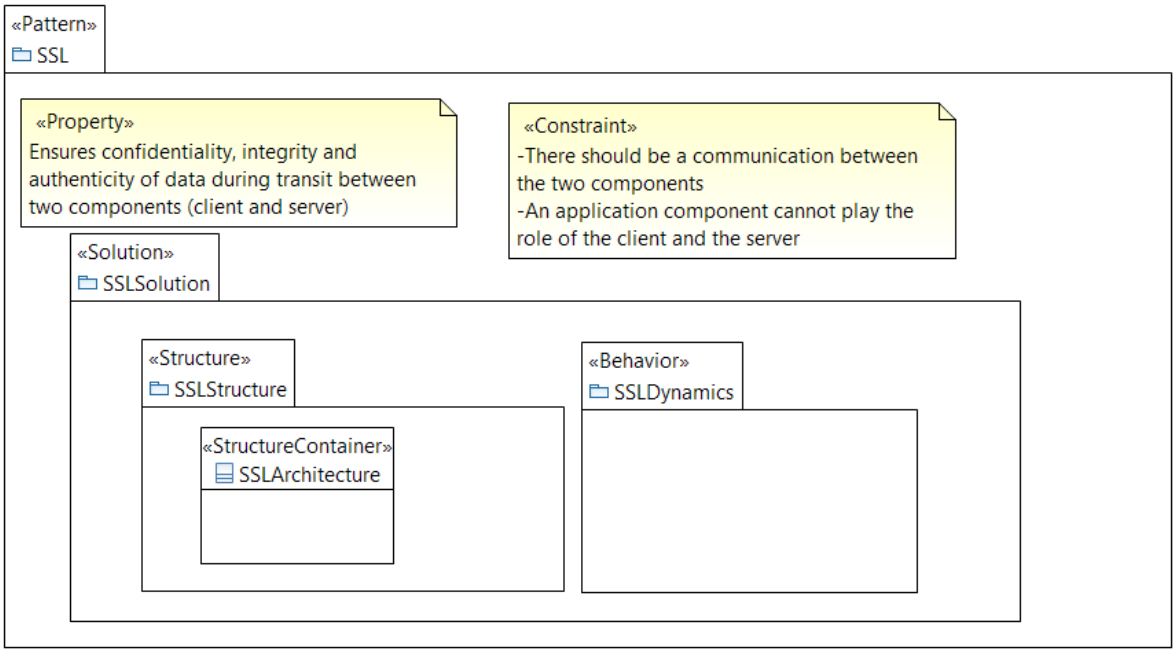


Figure 4.19: SSL Pattern instantiated SepmUML

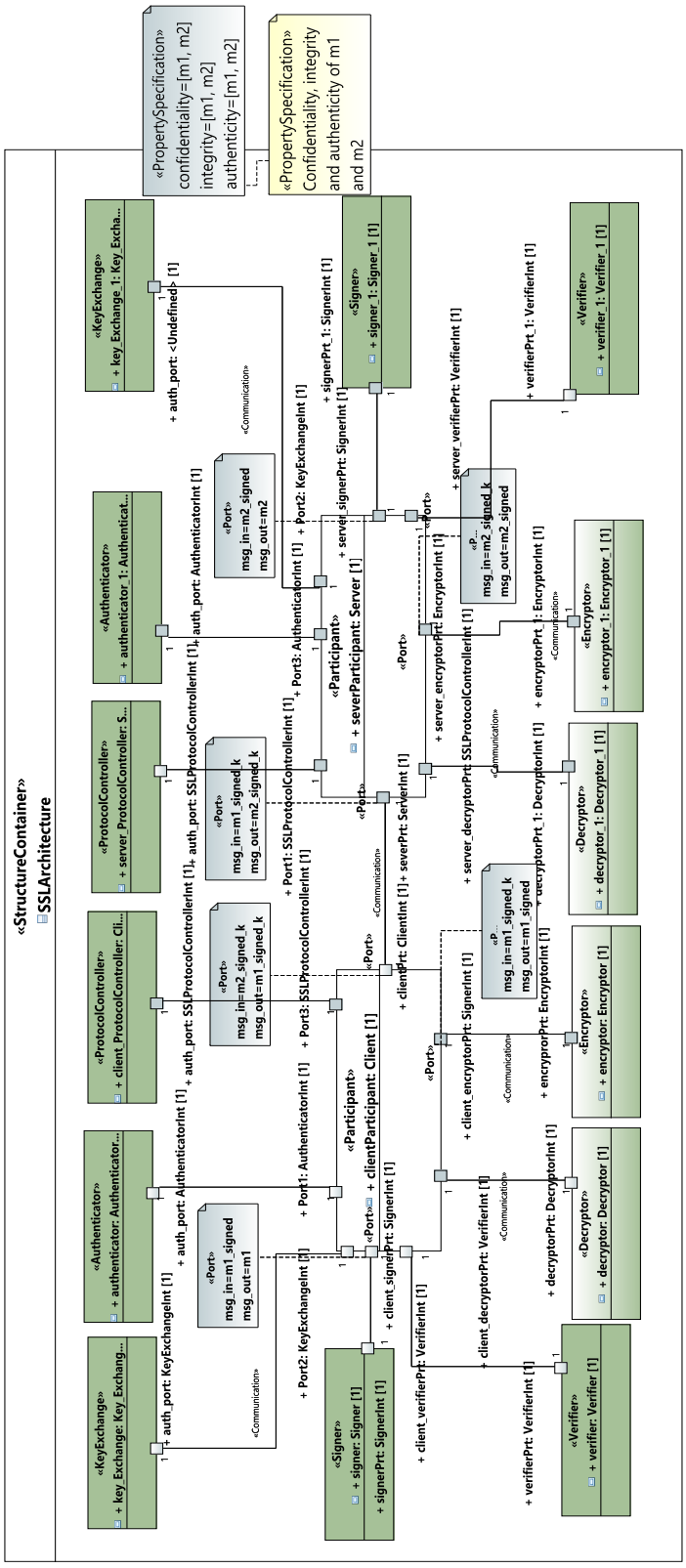


Figure 4.20: SSL Pattern Solution

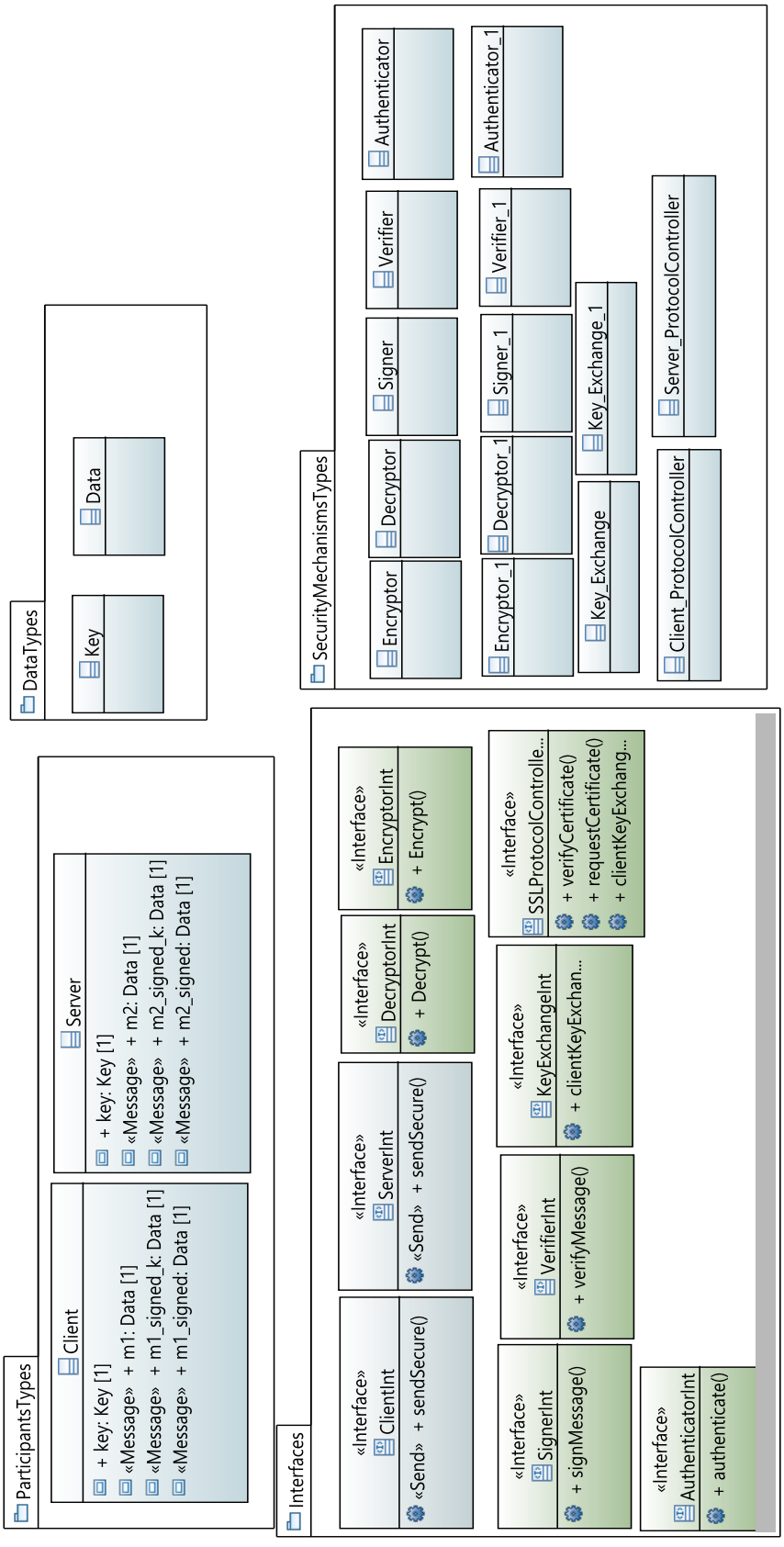


Figure 4.21: SSL Pattern Types and Interfaces

Pattern Solution

Figure 4.20 shows the solution of the SSL pattern. It has two pattern participants *clientParticipant* and *serverParticipant*. It contains the following security mechanisms:

- Protocol controller: main component that controls all the other security mechanisms to perform the *SSL Handshake* and *SSL record* protocols.
- Authenticator: authenticates client or server sides holding a certificate.
- KeyExchange: computes the client key exchange used to encrypt messages.
- Encryptor: encrypts transmitted messages.
- Decryptor: decrypts received messages.
- Signer: produces for each message a signature that guarantees authenticity and integrity of the message. It is sent together with the message.
- Verifier: verifies the integrity and authenticity of the message via its accompanied signature.

In addition, all the types and interfaces used to model the secure communication SSL pattern are represented in Figure 4.21. The pattern solution also contains transmitted messages.

Pattern Properties (Postconditions)

The specification of the properties is represented under the form of a comment stereotyped with *«PropertySpecification»* with the following attributes *confidentiality*, *integrity* and *authenticity*. As presented in Figure 4.20, the pattern provides the following properties:

- **Property 1.** confidentiality of messages m_1 and m_2.
- **Property 2.** integrity of messages m_1 and m_2.
- **Property 3.** authenticity of messages m_1 and m_2.

Pattern constraints (Preconditions)

Following the hypothesis in section 4.4.3. The preconditions are the following:

- **Precondition 1.** All pattern participants should be bound (one participant per component).
- **Precondition 2.** All communications in the pattern should exist in the application.

4.5.5 Integration: Elicitation

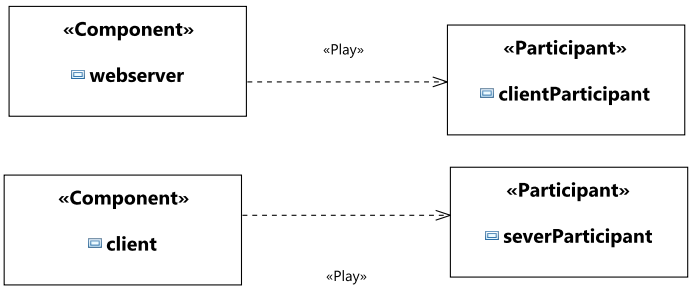


Figure 4.22: Application and SSL Secure Communication Pattern Casting diagram

In this phase, the casting diagram is established. Figure 4.22 shows the following bindings between the SSL patterns and components of the web application:

- b1: client *plays* role clientParticipant
- b2: server *plays* role serverParticipant

4.5.6 Integration: Context checking

In this phase, the casting diagram is validated against *Preconditions 1 and 2*. The validation of the castings against the preconditions is done using OCL invariants. Listing. 4.2 and Listing. 4.3 validate *Preconditions 1* and *Preconditions 2* respectively. In this case The preconditions are valid so we move to the next phase.

```
1 //Precondition 1 : All pattern participants should be bound (one participant per
      component)
2 Context Castings
3 self.play->select(p1,p2 |
4
5 (p1.participant = p2.participant
```

```
6 implies
7  p1.component = p2. participant)
8
9 and
10
11 (p1.component = p2. participant
12 implies
13 p1.participant = p2.participant)
14
15 and
16 p1.participant.structureContainter.participants->forAll(participant | self.play->exists(
     p_ | p_.participant = participant)))
```

Listing 4.2: Precondition 1: All pattern participants should be bound (one participant per component)

```
1
2 //Precondition 2: all communications in the pattern should exists in the application
3 Context Castings
4 self.play->forAll(p1,p2 |
5  (p1._'<>'(p2) and p1.participant.ports->exists(p1_in | p2.participant.ports->exists(
       p2_in | p1_in.communication = p2_in.communication)))
6 implies
7  p1.component.ports->exists(p1_in | p2.component.ports->exists( p2_in | p1_in.
       communication = p2_in.communication))
8 )
```

Listing 4.3: Precondition 2: all communications in the pattern should exist in the application

4.5.7 Integration: Merge

Here we detail the merge algorithm in Listing. 4.1 with the elements of SepmUML and ComponentUML. We also show its implementation using QVTo and the result of the phase for the working example.

Merge algorithm with modeling artifacts

In the merge phase, the goal is to merge the *StructureContainer* and the *application* based on *casting diagram*. The algorithm in Listing. 4.4 is a detailed version of the merge algorithm in Listing. 4.1. We add information about the used modeling elements of SepmUML and ComponentUML.

```
1
2  Algorithm Merge
3
4  modeltypes : SepmUML, ComponentUML
5
6  Input: A : ComponentUML::Application, Pattern : SepmUML::Pattern, C :
7  Set(SepmUML::Play).
8  Output: RA : ComponentUML::Application.
9
10 RA:= duplicate(A)
11 RA.addProperty(Pattern.structure.structureContainer.propertySpecification)
12 for each play : SepmUML::Play in C' do
13   component1 = play.component
14   patternParticipant1 = play.patternParticipant
15
16   for each pPort : ComponentUML::Port in patternParticipant1
17    if pPort.communication.ports.component->includes(c |
18    c.isKindOf(SepmUML::SecurityMechanism)) Port1 =
19    RA.duplicatePort(pPort,component1) securityMechanism =
20    RA.duplicateComponent(pPort.communication.ports.component->select(c|c.isKindOf(
          SecurityMechanism))
21    RA.CreateCommunication(Port1,securityMechanism.port) else
22     patternPartipant2 = pPort.communication.ports.component->select(PP | PP !=
23     patternParticipant1) component2 = patternParticipant.play.component
24     Port1 = RA.ports->select(input|portt.component = component1 and
25     Port1.communication.connects(component1,component2))
26     for each pOperation : ComponentUML::Operation in pPort.interface.operations
27      if Port1.interface.operations->includes(pOperation) == false
28       RA.addOperation(pOperation, Port1.interface)
29      endif
30     endfor
31
32    endif
33   endfor
34 endfor
```

Listing 4.4: Model-Based Merge algorithm

Implementing the algorithm using QVTo

Figure 4.23 shows an overview of the transformation rules using QVTo under EMF and presented in Listing. 4.4.

Illustration

In this phase, we perform the Model-To-Model transformation presented in section 4.5.7. The transformation takes as input, the application in Figure 4.8 and the castings in Figure 4.22. The Merge outputs a new application model in Figure 4.24. In addition,

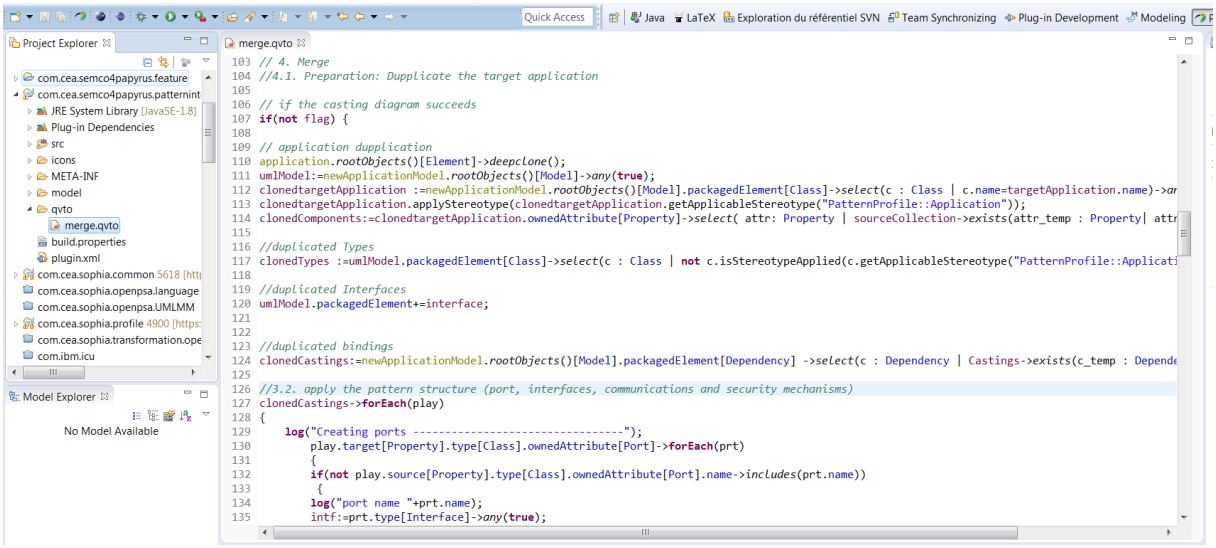


Figure 4.23: Merge phase with QVTo

Figure 4.25 shows that types and interfaces of the initial application have been modified adding new attributes and operations. In addition, new interfaces have been added relevant to security mechanisms.

4.5.8 Integration: Verification & Validation

At this stage, the new application verifies the postconditions. At each change (e.g., ad-hoc tailoring or the integration of another pattern), the application is validated against the postconditions. The validation of the application against the postconditions is done using the OCL invariant in Listing. 4.5. If the postconditions still hold, the change is accepted and committed. Else, the change is dismissed.

In the illustration, the application is validated against *Property 1* by verifying these statements :

- There must be one *Encryptor* and *Signer* mechanisms for each component sending messages m_1 and m_2
- Messages m_1 and m_2 must be encrypted before being sent.

```
1 //Postconditions validation
2 Context Application
3 self.base_class.ownedAttributes->select(c | c.isoclAsKind(Comment))->select(c |
4 c.isStereotypeApplied(PropertySpecification))->forAll( p |
5
```

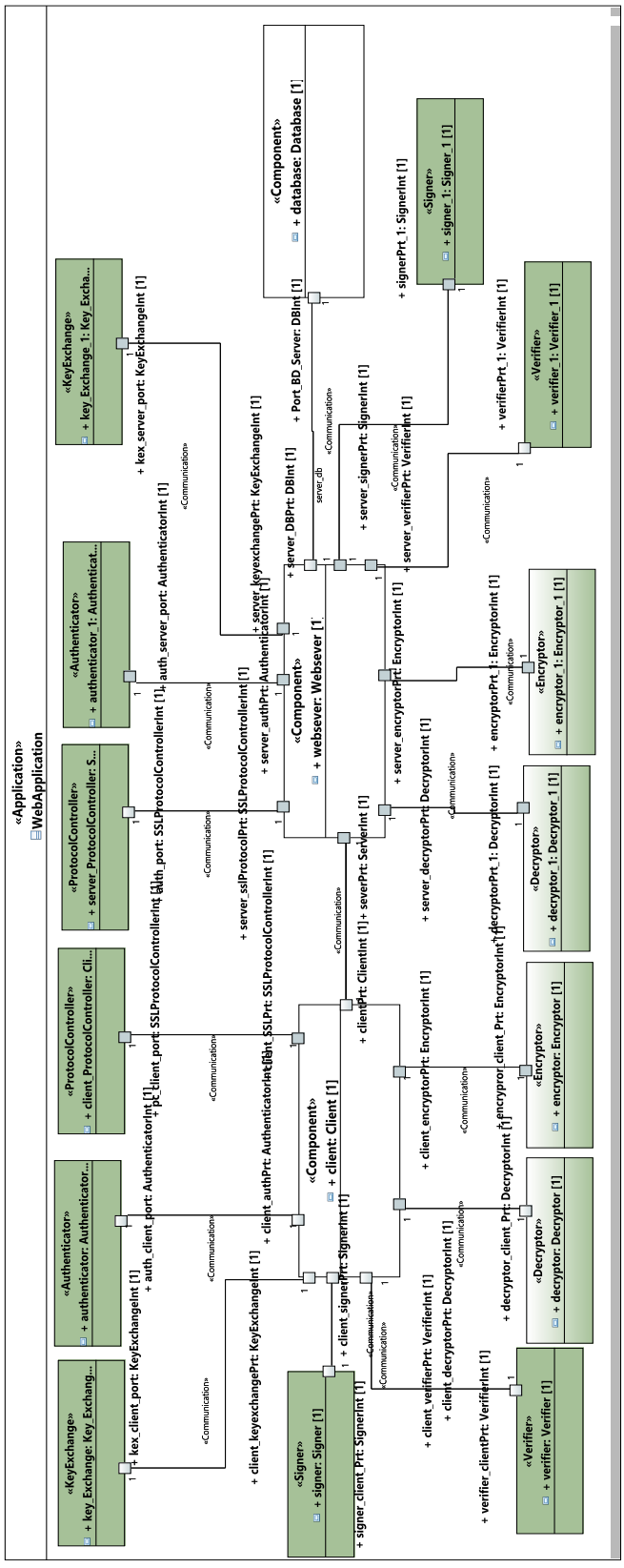


Figure 4.24: New Application diagram

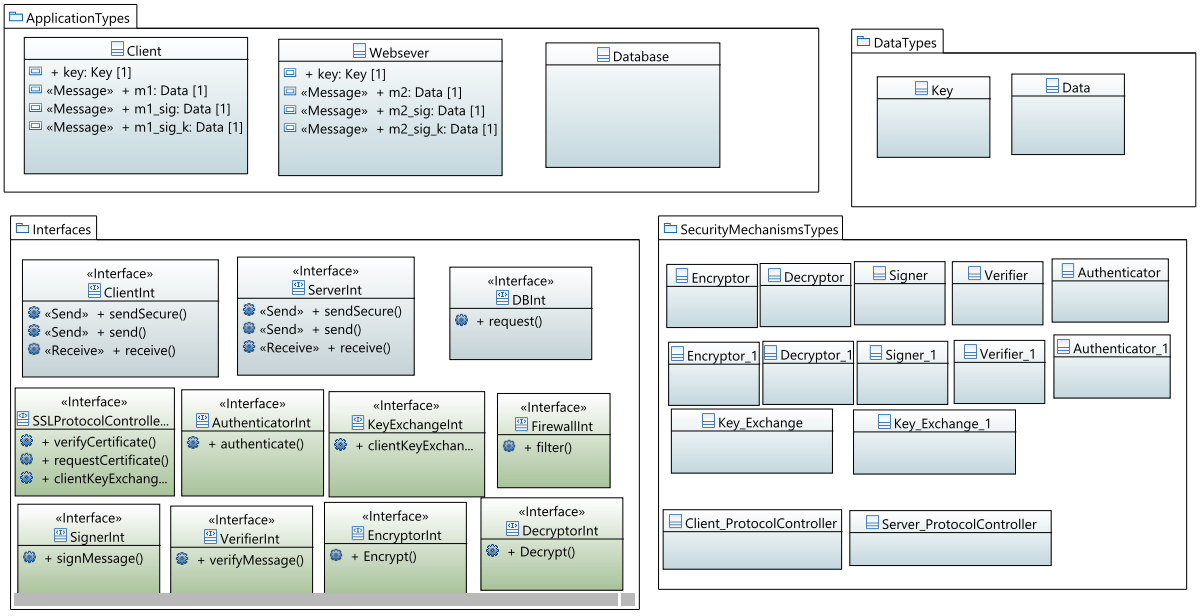


Figure 4.25: New Application Types (modified and added)

```
p.confidenlity->forAll( m_confidential |
// There exists a component that produces m_confidential and an ecryptor connected to this component that encrypts m_confidential
self.components->exists( c1|
c1.ports.msg_out = mconfidential
and
self.components->exists( enc|
enc.isOclKindOf(Encryptor)
and
c1.ports->exists(c1_in | enc.ports->exists(enc_in | c1_in.communication =
enc_in.communciation )))))

and

p.integrity->forAll( m_integrity |
// There exists a component that produces m_integrity and an signer connected to this component that signs m_integrity
self.components->exists( c1|
c1.ports.msg_out = m_integrity
and
self.components->exists( signer|
signer.isOclKindOf(Signer)
and
c1.ports->exists(c1_in | signer.ports->exists(signer_in | c1_in.communication =
enc_in.communciation )))))

and

p.authenticity->forAll( m_authenticity |
// There exists a component that produces m_authenticity and an signer connected to this component that signs m_authenticity
self.components->exists( c1|
c1.ports.msg_out = m_authenticity
and
self.components->exists( signer|
signer.isOclKindOf(Signer)
and
c1.ports->exists(c1_in | signer.ports->exists(signer_in | c1_in.communication =
enc_in.communciation )))))


)
```

Listing 4.5: OCL Queries for pattern property verification

4.6 Tool Support

We have rapidly described a global process for selecting, instantiating and integrating technical solutions capitalized in patterns in order to improve security of software architecture

designs. As we have seen, this takes several aspects and stages that must be supported by specific modeling and tooling. Indeed this approach requires first an improved pattern description formalism, second, the integration of dedicated merging techniques and finally verification facilities to ensure correct integration of these technical solutions. In the next section we refine the requirements for a convenient tooling support for this approach.

4.6.1 Tool support Requirements

The proposed tool chain is designed to support the proposed metamodels and Model-To-Model transformations. In order to support our approach, tools must fulfill the following key requirements:

- Enable the creation of the UML models used to describe system and software architecture.
- Allow the creation of a custom UML profile.
- Support the implementation of a repository to store pattern models and the related model libraries for classification and relationships.
- Enable the creation of visualizations of the repository to facilitate its access.
- Support the access to the repository. Create views on the repository according to its APIs, its organization and the needs of the targeted system engineering process.
- Enable transformations of the pattern models from the repository format into the target-modeling environment.
- Enable the creation of System of Patterns models in the target-modeling environment.
- Enable the creation of System of Patterns configuration models in the target-modeling environment.
- Enable the integration of application models and models imported from the repository.

In our case, we have chosen the following support tools :

- UML modeling environment: Papyrus[1] (Existing)

[1]https://eclipse.org/papyrus/

- Model Repository : SEMCOMDT[2] (SEMCO Model Development Tools, IRIT's editor and platform plugins) is used to support pattern repository (Existing).
- Selection, Instantiation and Integration of Pattern Models: Semco4Papyrus (Implemented)

4.6.2 Semco4Papyrus

Semco4Papyrus is an eclipse feature. It is based on the modeling environment *Papyrus* and composed of two plug-ins (modules):

- Access Tool providing:
  - GUI exposed to the end user
  - Selection of patterns stored in SEMCOMDT repository
  - Instantiation of patterns into Papyrus
- PatternIntegrator:
  - A tool support for correct pattern integration in Papyrus.
  - Provides a generic scheme for pattern integration instead of ad-hoc transformation. It is based on merge and verification techniques.

To install we provide an update site (see Figure 4.26) so that the user can select the needed modules.

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

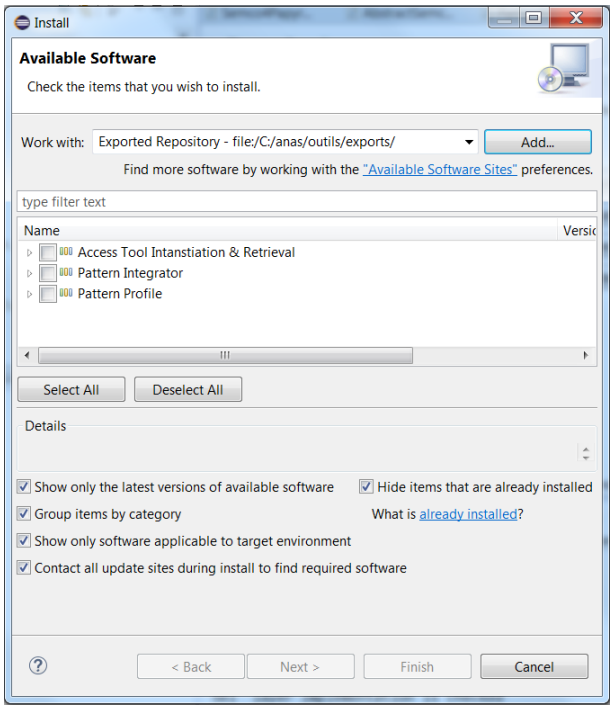


Figure 4.26: Semco4Papyrus eclipse update site

4.6.3 Access Tool

The access tool is a GUI for selecting patterns from SEMCOMDT and instantiating them into Papyrus. The GUI offers the possibility to search for DI and DS patterns at different life cycle stages. The GUI is composed of three areas

1. Search: the *basic search* is done by keyword and the *advanced search* by:
    - Domain (DI/DS)
    - Name
    - Threat Category (Man-In-The-Middle (MITM), Tampering, Injection, Denial of Service)
    - Security Property Category (confidentiality, integrity, authenticity, authentication, availability, non-repudiation)
    - Resource category
    - Life cycle stage (Requirements, Analysis, System Architecture, SW/HW architecture, Detailed design)
    - Architecture Layer

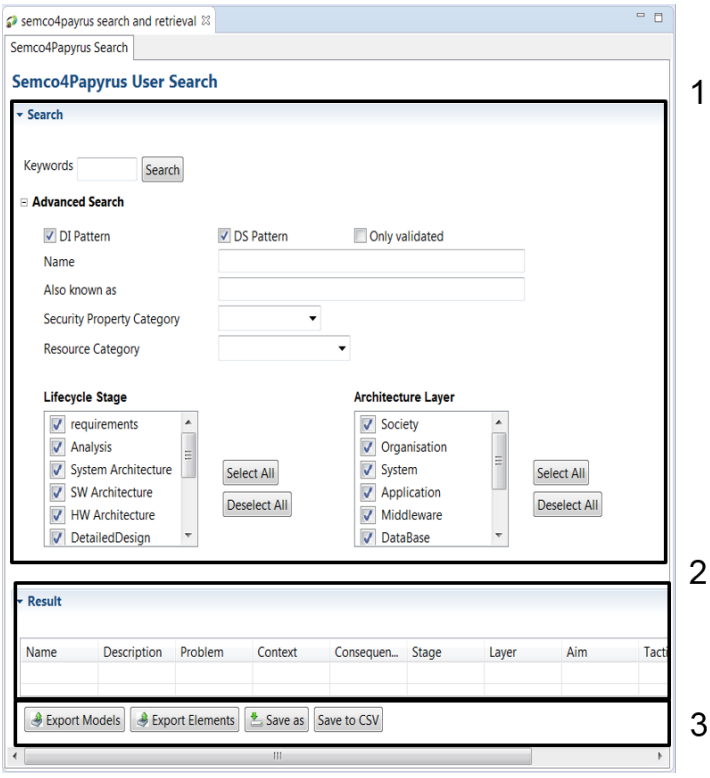


Figure 4.27: Semco4Papyrus access tool

2. Result: shows the result of the basic/advanced search. The results are a set of patterns (System of Patterns).

3. Retrieve Patterns into Papyrus: allows the retrieval of the selected System of Patterns into Papyrus.

4.6.4 Integration Module

PatternIntegrator is the integration module in charge of pattern model integration into the application. Figure 4.28 shows the commands responsible for the integration phases.

- Validate Preconditions: responsible for Phase 3
- Merge pattern: responsible for Phase 4
- Validate Postconditions: responsible for Phase 5

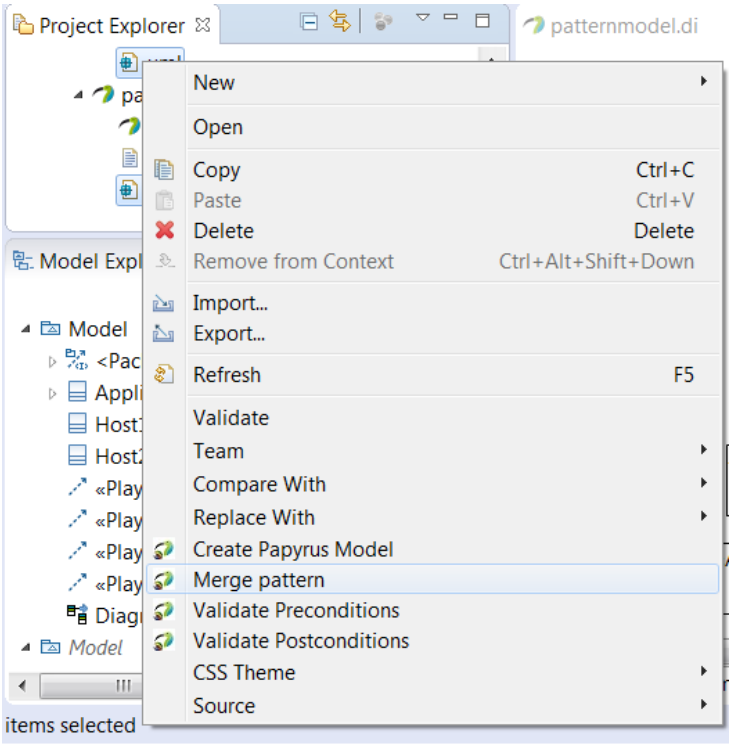


Figure 4.28: PatternIntegrator commands

4.7 Conclusion

In this chapter, we have proposed an approach and tool support for the selection, instantiation and integration of proper security patterns during architecture design based on risk assessment recommendations (to answer a part of: **RG1**, **RG2** and **RG4**). The modeling is based on accepted OMG standards (UML, its profile extension mechanism and OCL). The integration process is based on merge and OCL verification techniques using an MDE-based approach for security patterns. The integration process is composed of five phases: preparation, elicitation, context validation, Merge and Verification & Validation and offers to the user a flexible framework to integrate patterns. The pattern integration is a difficult task because all the elements of the pattern must be integrated in an application without compromising the system integrity and quality while guaranteeing the new properties introduced by the pattern. While most of works focus on Merging techniques for integrating patterns in an application, we pay more attention to the Verification & Validation phase to validate the integration.

The real added value of our approach is to make the validation of the integration in an easier way. On most of works, when the Verification & Validation is treated, the user must specify what property they should validate and generally in a specific formal language that the user is not familiar with. In our case, the properties are already part of the pattern, the user nearly needs to perform a binding in order to verify and validate the integration.

As a future work, we intend to use common Component-Based Modeling Languages e.g., Unified Component Model (UCM) [131], in order to increase acceptance of the approach. In addition, we also need to consider behavior during the integration process. Currently, we consider messages as part of the structure of the solution. Hence, messages should be represented in a dedicated diagram such as the UML information diagram that takes into account the order of messages.

In the next chapter, we discuss the analysis resulting software architecture candidates of the application according to residual threats. Indeed, the selected System of Patterns stops or mitigates specific system threats derived from risk assessment. However other system threats have been accepted. The goal of the next chapter is a holistic exploration of the software architecture of the application for the elicitation of unprotected flaws.

Chapter 5

Software Threat Analysis of Software Architectures

Contents

5.1 Introduction

Architecture threat analysis is very useful when it comes to detecting threats at early stages. Reported vulnerabilities show that architecture design weaknesses represent half of the total vulnerabilities of a system. Existing threat classification references can be used in that sense. However, the complexity of systems requires automated tool support.

In this chapter, we propose a scenario-based approach and its tool support for analyzing software architectures in order to allow automatic threat detection (to answer a part of: **RG1**, **RG3** and **RG4**). Software threat analysis at the level of the software architecture is a complement to risk assessment at the system architecture because software threats appear due to the level of details of the software architecture. Hence, risks identified by risks assessment are reevaluated and can be treated accordingly. We have formalized four threat scenarios namely: Man-In-the-Middle (MITM), Tampering, Injec-

tion and Denial of Service (DoS). We explain the formalization process through iterations. For each iteration, the soundness and completeness of the results are evaluated.

We recall the process described in Figure 3.1. Here, we focus on activity (A3.1). The remainder of the chapter is organized as follows. In section 5.2, we present related work tackling software architecture threat analysis and discuss the contributions of our work. In Section 5.3, we give an overview of the threat analysis process applied to a software architecture. We explain the threat scenarios formalization process using OCL through four iterations in section 5.4. In section 5.5, we give an overview of the tool support requirements and describe the used one. In section 5.6, we illustrate threat analysis on Microsoft's web application working example after the application of SSL pattern. In section 5.7, we conclude and sum up the contributions.

5.2 Related work

Several efforts have focused on assessing quality attributes in software architectures such as modifiability, portability, maintainability, etc. The Architecture Trade-off Analysis Method (ATAM) [87] and the Software Architecture Analysis Method (SAAM) [86] are well-known methods for performing assessment of quality attributes. Hence some approaches extended these methods in order to assess security. Architecture assessment approaches can be categorized into two groups: scenario-based and property-based approaches.

5.2.1 Scenario-based Analysis

Scenario-based approaches consist on modeling security scenarios and then analyzing the architecture with regards to these scenarios. These approaches are generally refinements of ATAM and SAAM. In literature, most of these works [10, 9, 156, 22] have limitations in formalizing scenarios, in reusing and extending them, in automatizing the verification process and they also lack tool support.

Modeling threat scenarios is based on experience in the security domain. Threat scenario classifications (e.g., STRIDE, Common Attack Pattern Enumeration and Classification (CAPEC) [1] and Common Weakness Enumeration (CWE) [2]) help understanding how attackers may exploit flaws in the architecture. The problem is that these references, although interesting and useful are not formalized. To this end, some works have used OCL language to specify signatures of these threats to automatically detect threats in the software architecture. In [156], the authors propose a framework for detecting architec-

tural flaws over a code. It starts by generating a graph describing a run-time architecture using static analysis. Then they assign security properties on the graph of objects. The process is incremental and semi-automatic since the architect gains knowledge about the software architecture by querying and annotating the objects of the graph with security properties such as trust, criticality, etc. The architect defines a security policy as a set of constraints over sets returned by queries. The constraints in this approach are highly dependent on the application and are not generic or reusable. In our approach we aim at fostering reuse. In [22], the authors present a framework for detecting flaws in the code. The code is first transformed in STRIDE Data Flow Diagrams (DFDs) using static analysis. Then based on a 'best practice' repository where threat patterns are stored, an automatic check is performed to detect the threats and security measures that may be applied as annotations to DFDs to mitigate these threats. SecureUML [99] is a modeling language for specifying access control requirements in terms of declarative aspects based on Role-based Access Control (RBAC) but extends the latter with authorization constraints to specify dynamic properties in terms of programmatic aspects. Basin et al. [19] use a metamodel called SecureUML+ComponentUML that combines SecureUML [99] 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. The two artifacts are analyzed by evaluating OCL queries. The evaluation serves on one hand to detect and correct design flaws, if there are any, in the security design model or in the security scenario. On the other hand it allows having information about the allowed accesses of each user. SecureUML is specifically designed for evaluating the authorizations of the access controls of an application whereas our approach evaluates if the architecture has the necessary mechanisms to mitigate threats. In that sense, the two approaches are complementary. Other works do not formalize security scenarios such as [9]. In this work, Alkussayer et al. report a security scenario-based and risk-based evaluation framework for assessing software architecture. The process generates security scenarios and evaluates threats. If the results are unsatisfactory then a set of security patterns are integrated to mitigate them.

5.2.2 Property-Based Analysis

Property-based approaches focus on formalizing security properties to assess a software architecture Antonino et al. [15] presented a method for analyzing security in Service Oriented Architectures (SOA) named SiSOA. The method exploites reverse engineering to extract security-relevant facts. They then use system-independent indicators and a

knowledge base which stores a set of rules consisting of security goals and indicators relevant for every goal. The approach is extensible but not fully automated since it requires an expert to analyze the architecture according to system-related information and the rules stored in the knowledge base. Santanna et al. [138] proposed an approach to measure the modularity of cross-cutting concerns at architectural level. They defined a set of modularity properties used for analyzing the architecture. Recently, Hamid et al. [65] proposed a modeling language for the specification of security patterns. In addition, the authors provided a formal language and a validation mechanism to enable the verification and security properties.

5.2.3 Property and Scenario-Based Analysis

Property-based and Scenario-based belong to both categories of approaches. A forefront approach is the work of [85]. Jürjens defines a UML profile that authorizes the expression of information related to security using UML diagrams. The UMLsec profile has been established using three mechanisms of UML extensions which are stereotypes, tagged values attached to the stereotypes and constraints. These mechanisms define generic security properties (Confidentiality, Integrity, Data flow security, Access control, Auditability and traceability). These properties are verified during the analysis of the design against an adversary model. In [10], Almorsy et al. have formalized using OCL four threat scenarios (Denial of Service, Tampering, Injection and Man-In-The-Middle) and a set of properties that measure the attack surface, compartmentalization, least privilege and fail securely. The software architecture is modeled using UML called System Description Model (SDM). It is then mapped to the Security Specification Model (SSM) that represents security mechanisms. Our added value is that the analysis does not only check if the needed security mechanisms exist but also that they are correctly used to mitigate the threat scenarios.

5.2.4 Positioning

Our approach is a scenario-based approach. We have formalized a number of threat scenarios. Table 5.1, compares our approach to the aforementioned ones mainly: Almorsy et al. [10], Vanciu et al. [156] and Berger et al. [22]. A scenario-based analysis approach has to:

- Foster reuse of the formalized threat scenario *(Criteria 1)*.
- Verify that the architecture has the right security mechanisms *(Criteria 2)*.

- Verify that these security mechanisms are used correctly *(Criteria 3)*.
- Have a list of well-known threat scenarios. *(Criteria 4)*.

Approaches	*(Criteria 1)*	*(Criteria 2)*	*(Criteria 3)*	*(Criteria 4)*
Our approach	✓	✓	✓	DoS, MITM, Tampering, Injection
Almorsy et al. [10]	✓	✓	×	DoS, MITM, Tampering, Injection
Vanciu et al. [156]	×	✓	✓	Information disclosure,Tampering
Berger et al. [22]	✓	✓	×	Information Disclosure

Table 5.1: Positioning of our contribution with regards to scenario-based approaches

5.3 Threat analysis based on formalized threat scenarios

We propose an approach for detecting threats in the software architecture based on a set of formalized threat scenarios.

5.3.1 Methodology description

The approach depicted in Figure 5.1 allows the analysis of software architectures in order to detect existing threats according to formalized threat scenarios. The first step consists in formalizing threat scenarios using OCL from existing threat classification references **(step 0)**. Once these threat scenarios are formalized, the software architecture model, obtained by security pattern integration, is passed to the analysis module **(step 1)**. This step outputs existing threats.

5.3.2 Step 0: Threat scenarios formalization

Specifying threat scenarios is based on experience in the security domain thus this activity should be done by security experts. Once formalized, these threat scenarios are stored in a knowledge base.

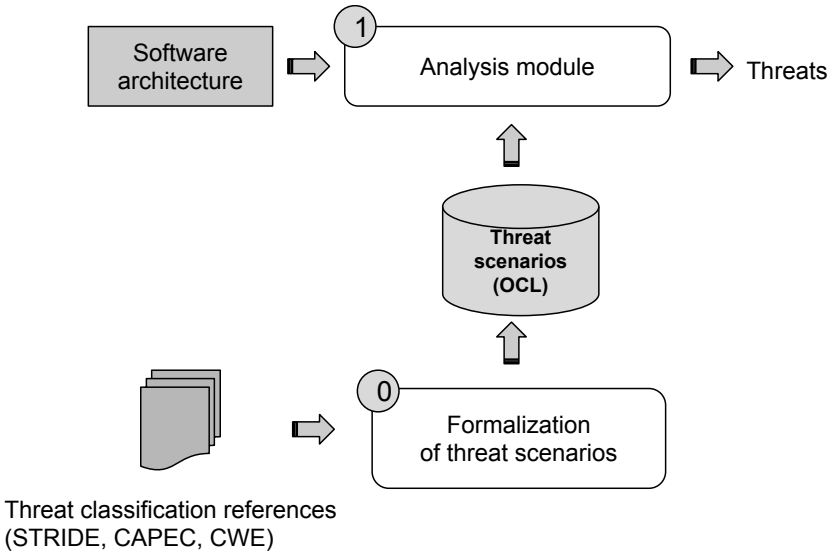


Figure 5.1: Threat Analysis Process

Inputs during our formalization process are existing threat classifications: STRIDE [144], Common Attack Pattern Enumeration and Classification (CAPEC) [1], Common Weakness Enumeration (CWE)[2]. These references describe informally a set of threat scenarios. Each threat scenario has a signature. This signature specifies the conditions in which a threat can occur. Thus it defines the threats according to a certain scenario. However, the threat scenarios are described informally and thus applying them manually is error-prone and time consuming. Formalizing threat scenarios allows the automation of security architecture threat analysis by detecting existing threats. We discuss below the considered threat scenarios that we have taken from the classifications discussed above. This is neither a comprehensive nor a complete list but we tried to cover well-known categories frequently used. For instance, the injection category includes SQL injections, OS command injections and XPath injections and is ranked number 1 in OWASP top 10:

- Man-In-The-Middle (MITM): is responsible for relaying or altering messages between two parties. The signature of this threat scenario is: lack and/or weakness of encryption and Authenticity mechanisms.

- Denial of Service (DoS): can make the system resources unavailable for authorized users. The signature of this threat scenario is: lack and/or weakness of Firewall, Authentication and Authorization mechanisms.

- Tampering: is responsible for altering data at storage or during transmission. The

signature of this threat scenario is: lack and/or weakness of Authenticity mechanisms.

- Injection: is responsible for passing malicious inputs in order to gain higher privileges, alter data, or crash the system. The signature of this threat scenario is: lack and/or weakness of Firewall, Authentication and Authorization mechanisms.

We use OCL language [126] to formalize the aforementioned threat scenarios as OCL invariants. These invariants are stored and used by the analysis module in the next step.

5.3.3 Step 1: Analysis module

The analysis module is responsible for analyzing the software architecture model. If an invariant is violated then there is a threat relevant to the threat scenario.

5.4 Formalizing threat scenarios using OCL

To illustrate the formalization process presented in the next section of this chapter, we use the web application working example in Figure 2.11 presented in section 2.6.1. This architecture is deployed on a platform described hereunder in Figure 5.2. Table 5.2 describes the deployment of software components on the platform nodes.

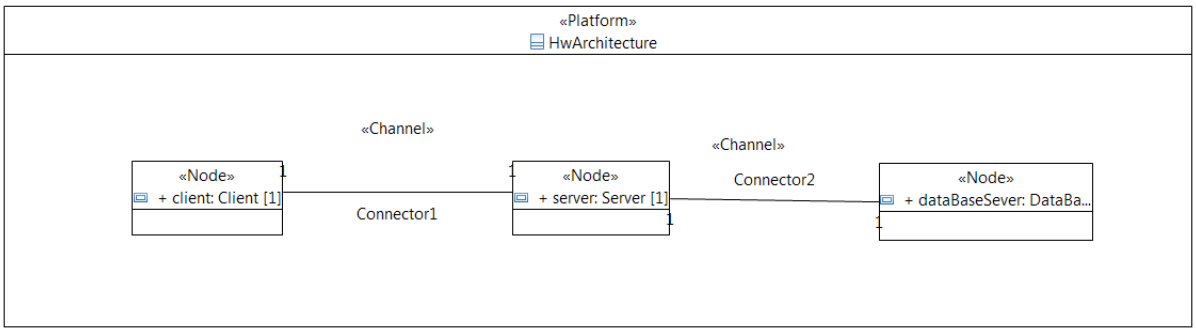


Figure 5.2: Web application platform

We enumerate the list of software threats using Microsoft threat modeling process [110] for the web application and their category in Table 5.3.

In order to exploit these results during the formalization process, we give the number of threats for each threat category and the total number of threats in the web application shown in Table 5.4.

Software Components	Hardware Nodes
client (browser)	client
website	web server
database	database server

Table 5.2: Deployment of the software component on the hardware nodes

Vulnerabilities	Threat category
Using poor input validation that leads to cross-site scripting (XSS), SQL injection, and buffer overflow attacks.	Denial of Service (DoS), Injection
Passing authentication credentials or authentication cookies over unencrypted network links, which can lead to credential capture or session hijacking.	Man-In-The-Middle (MITM)
Using weak password and account policies, which can lead to unauthorized access.	Elevation of Privilege, Denial of Service (DoS)
Failing to secure the configuration management aspects of your application, including administration interfaces.	Denial of Service (DoS)
Storing configuration secrets, such as connection strings and service account credentials, in clear text.	Information disclosure
Using over-privileged process and service accounts.	Elevation of privilege
Using insecure data access coding techniques, which can increase the threat posed by SQL injection.	Injection
Using weak or custom encryption and failing to adequately secure encryption keys.	Information Disclosure
Relying on the integrity of parameters that are passed from the Web browser, for example, form fields, query strings, cookie data, and HTTP headers.	Tampering
Using insecure exception handling, which can lead to Denial of Service (DoS) attacks and the disclosure of system-level details that are useful to an attacker.	Denial of Service (DoS), Information disclosure
Doing inadequate auditing and logging, which can lead to repudiation threats.	Repudiation

Table 5.3: Web application vulnerabilities and their threat categories [110]

Threat scenario	Number of threats
Man-In-The-Middle	2
Tampering	2
Denial of Service	2
Injection	2
Total	8

Table 5.4: Number of threats per scenario

The main objective of this work is to analyze software architectures allowing the detection of threats according to formalized threats. In this section, we describe the iterations that allowed detecting threats. This has required the addition of threats analysis concepts in ComponentUML. In addition we have formalized at each iteration threat scenarios using OCL. We show, at each iteration, the improvement of the soundness and completeness of the results with regards to the web application working example:

- Iteration 1: It starts with the initial ComponentUML. In this iteration we consider only Man-In-The-Middle (MITM) and Tampering threat scenarios. For each threat scenario, we explore the software architecture to check if the necessary cryptographic mechanisms exist for two components that are deployed into two different nodes.

- Iteration 2: In this iteration, we verify not only the existence of security mechanism but that they are correctly used i.e., we verify that transmitted messages between two components are encrypted and/or signed. We add the concept of trust boundary i.e., we check if the component is deployed in a trusted or untrusted zone.

- Iteration 3: In this iteration, we consider Denial of Service (DoS) and Injection threat scenarios.

- Iteration 4: In this iteration, we add the concept of port kind i.e., if the port is external (public) or internal (private).

5.4.1 Iteration Evaluation Metrics

In order to evaluate the formalized threat scenarios, we use a set of evaluation metrics [134] to measure at each iteration the soundness and completeness of the detection results.

- **Precision rate** measures the soundness of the results. A high precision rate means that the detected threats contain more True Positives (TP) i.e., valid results than

False Positives (FP) i.e., false results. It is computed as follows:

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

- **Recall rate** measures the completeness of the results. A high recall rate means that the detected threats give more valid results than misses them (False Negatives (FN)). It is computed as follows:

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

- **F-Measure** measures the soundness and completeness of the results with weights for each metric. We assume that the two metrics have equal weights. It is computed as follows:

$$F - Measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.3}$$

5.4.2 Iteration 1

In this iteration, we start with ComponentUML in Figure 4.3 and we consider only Man-In-The-Middle (MITM) and Tampering threat scenarios. For each threat scenario, we explore the software architecture to check that the necessary cryptographic mechanisms exist for two components that are deployed into two different nodes.

Threat scenarios formalization

We formalize Man-In-The-Middle (MITM) and Tampering threat scenarios using OCL. In order to do this we inspect the threats relevant to these scenarios:

- Man-In the middle threats exploit the lack of encryption and integrity protection mechanisms
- Tampering threat scenarios exploit the lack of integrity protection mechanisms

Based on the previous, in Listings 5.1 and 5.2 are given these OCL constraints of Man-In-The middle and Tampering threat scenarios.

```
1 Context Application inv Man-In-The-Middle_v1
2 self.components->select(c1 |
3 self.components->exists(c2 |
4 not(c1.oclIsKindOf(PatternProfile::SecurityMechanism))
5 and
6 not(c2.oclIsKindOf(PatternProfile::SecurityMechanism))
```

```
7 and
8 /* if c1 and c2 are different*/
9 c1._'<>'(c2)
10 and
11 /* if c1 and c2 are deployed in different nodes*/
12 (c1.node)._'<>'(c2.node)
13 and
14 c1.node.channels->exists(ch | c2.node.channels->includes(ch))
15 and
16 /* c1 and c2 communicate */
17 (c1.ports->exists(inp| c2.ports->exists(inpt2 | inpt2.communication = inp.communication))
     )
18  and
19
20 /* The security mechanisms exist: encryptor, decryptor, signer and verifier */
21
22 (self.components->select(enc |  self.components->exists(dec , mac1 | self.components->
     exists(mac2|
23
24 enc.oclIsKindOf(PatternProfile::Encryptor)
25 and
26  dec.oclIsKindOf(PatternProfile::Decryptor)
27 and
28  mac1.oclIsKindOf(PatternProfile::Signer)
29 and
30  mac2.oclIsKindOf(PatternProfile::Verifier)
31 and
32  (enc.node = c1.node)
33 and
34   (dec.node = c2.node)
35 and
36   (mac1.node = c1.node)
37 and
38   (mac2.node = c2.node)
39 ))))))->size()
```

Listing 5.1: Man-In-The-Middle (MITM) threat scenario formalized using OCL

```
1 Context Application inv Tampering_v1
2
3 self.components->select(c1 |
4 self.components->exists(c2 |
5 not(c1.oclIsKindOf(PatternProfile::SecurityMechanism))
6 and
7 not(c2.oclIsKindOf(PatternProfile::SecurityMechanism))
8 and
9 c1._'<>'(c2)
10 and
11 (c1.node)._'<>'(c2.node)
12 and
13 c1.node.channels->exists(ch | c2.node.channels->includes(ch))
14 and
15 /* c1 and c2 communicate */
16
```

```
17 (c1.ports->exists(inp| c2.ports->exists(inpt2 | inpt2.communication = inp.communication))
       )
18 and
19 /* The security mechanisms exist: signer and verifier */
20
21 (self.components->exists(mac1,mac2 |
22   mac1.oclIsKindOf(PatternProfile::Signer)
23 and
24   mac2.oclIsKindOf(PatternProfile::Verifier)
25 and
26   (mac1.node = c1.node)
27 and
28   (mac2.node = c2.node)))))->size()
```

Listing 5.2: Tampering threat scenario formalized using OCL

Threat Analysis Results

Threat categories	**Detected Threats**	**TP**	**FP**	**FN**
Man-In-The-Middle	3	2	1	0
Tampering	3	2	1	0
Denial of Service	0	0	0	2
Injection	0	0	0	2
Total	6	4	2	4

Table 5.5: Number of threats per scenario

After checking OCL constraints over the working example, we obtain the results in Table 5.5.

Evaluation Metrics Results

From the threat analysis results, we compute *Precision*, *Recall* and *F-Measure* metrics. The results are shown in Table 5.6. The actual version of the threat scenarios formalized with OCL has a Precision, Recall and F-Measure rates of 66,66%, 50% and 57,14%. The precision rate, that measures soundness, means that more than half of the detected threats are FNs i.e, not threats. In order to increase the precision we must consider trust boundaries. Indeed, server and database have been considered trusted during Microsoft's threat modeling in Figure 2.11.

Metrics	**Values**(%)
Precision	66,66
Recall	50
F-Measure	57,14

Table 5.6: Evaluation metrics results

5.4.3 Iteration 2

In this iteration, We add the concept of trust boundary i.e., we check if the component is deployed in a trusted or untrusted zone. Figure 5.3 shows ComponentUML model with the *TrustLevel* enumeration with two literals *trusted* and *untrusted*. In addition, we verify not only the existence of security mechanisms but that they are correctly used. This is done by verifying that the transmitted messages between two components are encrypted and/or signed.

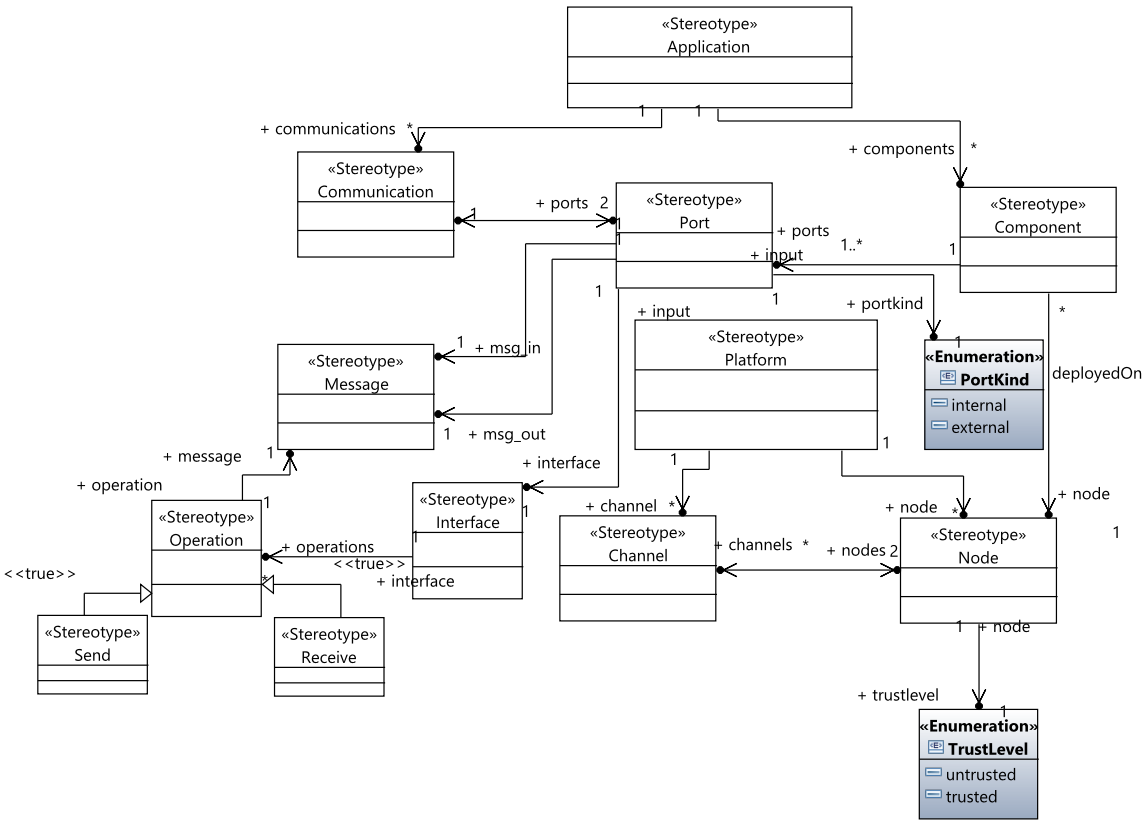


Figure 5.3: Augmented ComponentUML model

Threat scenarios formalization

We consider trust level of components and obtain a new version of the OCL constraints. In Listing 5.3 and Listing 5.4 are given extracts of these OCL constraints. Note that a full listing of the threat scenarios can be found in Appendix B.

```
1  Context Application inv Man-In-The-Middle_v2
2  self.components->select(c1 |
3
4  self.components->exists(c2 |
5
6  /* c1 or c2 are deployed in an untrusted node */
7  (c1.node.trustlevel= TrustLevel::untrusted or c2.node.trustlevel = TrustLevel::untrusted
       )
8  and
9
10 /* c1 and c2 two components commnunicating deployed in different nodes
11 [...]
12
13 and
14 /* The security mechanisms exist: encryptor, decryptor, signer and verifier */
15
16 [...]
17 and
18 (
19
20 /* Mecanisms are connected to components */
21
22 (c1.ports->exists(inp_c1 |  enc.ports->exists(inp_enc | inp_c1.communication = inp_enc.
       communication and inp_enc.communication._'<>'(null)))
23      and
24  c1.ports->exists(inp_enc |  mac1.ports->exists(inp_mac1 | inp_mac1.communication =
        inp_enc.communication and inp_enc.communication._'<>'(null)))
25  and
26  c2.ports->exists(inp_mac1 |  mac2.ports->exists(inp_mac2 | inp_mac2.communication =
        inp_mac1.communication and inp_mac1.communication._'<>'(null)))
27  and
28  c2.ports->exists(inp_mac2 |  dec.ports->exists(inp_dec | inp_mac2.communication =
        inp_dec.communication and  inp_dec.communication._'<>'(null)))
29  and
30
31 /* the mechanisms are called correctly */
32 c1.ports->select(inp_c1_c2 | inp_c1_c2.msg_out._'<>'(null) and (c1.ports->exists(inp| c2.
       ports->exists(inpt2 | inpt2.communication = inp_c1_c2.communication))))->forAll(
       inp_c1_c2 | mac1.ports->exists(sign_in |c1.ports->exists(c1_inp| c1_inp.communication
        = sign_in.communication  )
33  and
34  (enc.ports->exists(enc_in | c1.ports->exists(c1_signorEnc |
35 -- case 1: message flow sign and then encrypt
36       (c1_signorEnc.communication = enc_in.communication
37       and
38       sign_in.msg_out = inp_c1_c2.msg_out
39       and
40       sign_in.msg_in = enc_in.msg_out
```

```
41      and
42      enc_in.msg_in = c1_signorEnc.msg_out)
43           or
44 -- case 2: message flow encrypt and then sign
45      (c1_signorEnc.communication = sign_in.communication
46      and
47      enc_in.msg_out = inp_c1_c2.msg_out
48      and
49      enc_in.msg_in = sign_in.msg_out
50      and
51      sign_in.msg_in = c1_signorEnc.msg_out))))))))))->isEmpty()
52      )))->size()
```

Listing 5.3: Man-In-The-Middle (MITM) threat scenario version 2 formalized using OCL

```
1 Context Application inv Tampering_v2
2
3 self.components->select(c1 |
4
5 self.components->exists(c2 |
6
7 /* c1 or c2 are deployed in an untrusted node */
8 (c1.node.trustlevel= TrustLevel::untrusted or c2.node.trustlevel = TrustLevel::untrusted
      )
9 and
10
11 /* c1 and c2 two components communicating deployed in different nodes
12 [...]
13
14 and
15
16 /* case1: The security mechanisms exist: signer and verifier */
17 self.components->select(signer |
18 signer.oclIsKindOf(Signer)
19 and
20 signer.node = c1.node
21 and
22 self.components->exists(verifier |
23 verifier.oclIsKindOf(Verifier)
24  and
25 verifier.node = c2.node
26
27 /* Mechanisms are connected to components */
28 and
29 c1.ports->exists(inp_mac1 |  signer.ports->exists(inp_mac2 | inp_mac2.communication =
      inp_mac1.communication and inp_mac1.communication._'<>'(null)))
30 and
31 c2.ports->exists(inp_dec |  verifier.ports->exists(inp_c2 | inp_c2.communication =
      inp_dec.communication and  inp_dec.communication._'<>'(null))))
32 and
33
34 /* the mechanisms are called correctly */
35
36 c1.ports->select(inp_c1_c2 | inp_c1_c2.msg_out._'<>'(null) and (c1.ports->exists(inp| c2.
```

```
    ports->exists(inpt2 | inpt2.communication = inp_c1_c2.communication))))->forAll(
    inp_c1_c2 |
signer.ports->exists(sign_in |
c1.ports->exists(c1_inp| c1_inp.communication = sign_in.communication  )
and
c1.ports->exists(c1_sign |

(c1_sign.communication = sign_in.communication
and
sign_in.msg_out = inp_c1_c2.msg_out
and
c1_sign.msg_out = sign_in.msg_in
 )))))->isEmpty()

and

/* case2: The security mechanisms exist: signer and verifier and encryption
mechanisms */
[...]

 )))->size()
```

Listing 5.4: Tampering threat scenario version 2 formalized using OCL

Threat Analysis Results

Threat categories	**Detected threats**	**TP**	**FP**	**FN**
Man-In-The-Middle	2	2	0	0
Tampering	2	2	0	0
Denial of Service (DoS)	0	0	0	2
Injection	0	0	0	2
Total	4	4	0	4

Table 5.7: Number of threats per scenario

After checking the OCL constraints over the working example, we obtain the results in Table 5.7. The results show that four threats have been detected and are TPs.

Evaluation Metrics Results

From threat analysis results, we compute *Precision*, *Recall* and *F-Measure* metrics. The results are shown in Table 5.8. The second version of Man-In-The-Middle (MITM) and Tampering threat scenarios formalized with OCL has a Precision, Recall and F-Measure rates of 100%, 50% and 66,66% respectively. The precision rate, that measures soundness,

has increased to its maximum because all the detected threats are TP. The recall rate is still the same because Denial of Service (DoS) and Injection threat scenarios have not yet been formalized.

Metrics	**Values**(%)
Precision	100
Recall	50
F-Measure	66,66

Table 5.8: Evaluation metrics results

5.4.4 Iteration 3

In this iteration, we consider Denial of Service (DoS) and Injection threat scenarios.

Threat scenarios formalization

We formalize Denial of Service (DoS) and Injection threat scenarios using OCL. In order to do so, threats relevant to each scenario are inspected. Denial of Service and Injection threat scenarios exploit the lack of firewall, authentication and authorization mechanisms. In Listings 5.5 and 5.6 are given these OCL constraints. Note that a full listing of the threat scenarios can be found in Appendix B.

```
1 Context Application inv DenialofService_v1
2 self.components->select(c1 |
3 c1.ports->exists(port |
4
5 /* Firewall, authenticator, and authorizer mechanisms exist*/
6 self.components->select(firewall |
7 firewall.oclIsKindOf(Firewall)
8 and
9 firewall.node = c1.node
10 and
11 self.components->exists(auth |
12 auth.oclIsKindOf(Authenticator)
13 and
14 auth.node = c1.node
15 and
16 self.components->exists(authorizer |
17 authorizer.oclIsKindOf(Authorizer)
18 and
19 authorizer.node = c1.node
20 and
21
22 /*Security mechanisms are connected to component c1 and message flow is correct*/
23
24 --firewall and c1 are connected and input messages go from c1 to firewall --
```

```
25 c1.ports->exists(in_c1_firewall |  firewall.ports->exists(inp_firewall |
26 [...]
27
28 and
29 --authenticator and c1 are connected and input messages go from firewall to authenticator
       --
30 c1.ports->exists(in_c1_auth |  auth.ports->exists(inp_auth|
31 [...]
32
33 and
34 --authorizer and c1 are connected and input messages go from firewall to
35 authenticator -- c1.ports->exists(in_c1_authorizer |  authorizer.ports->exists(
       inp_authorizer|
36 [...]
37
38 )))))))))->size()
39
40 ))->isEmpty()
```

Listing 5.5: Denial of Service (DoS) threat scenario formalized using OCL

```
1 Context Application inv Injection_v1
2
3 Context Application inv DenialofService_v1
4 self.components->select(c1 |
5 c1.ports->exists(port |
6
7 /* Firewall, authenticator, and authorizer mechanisms exist*/
8 self.components->select(firewall |
9 firewall.oclIsKindOf(Firewall)
10 and
11 firewall.node = c1.node
12 and
13 self.components->exists(auth |
14 auth.oclIsKindOf(Authenticator)
15 and
16 auth.node = c1.node
17 and
18 self.components->exists(authorizer |
19 authorizer.oclIsKindOf(Authorizer)
20 and
21 authorizer.node = c1.node
22 and
23
24 /*Security mechanisms are connected to component c1 and message flow is correct*/
25
26 --firewall and c1 are connected and input messages go from c1 to firewall --
27 c1.ports->exists(in_c1_firewall |  firewall.ports->exists(inp_firewall |
28 [...]
29
30 and
31 --authenticator and c1 are connected and input messages go from firewall to authenticator
       --
32 c1.ports->exists(in_c1_auth |  auth.ports->exists(inp_auth|
```

```
33 [...]
34
35 and
36 --authorizer and c1 are connected and input messages go from firewall to
37 authenticator -- c1.ports->exists(in_c1_authorizer |  authorizer.ports->exists(
       inp_authorizer|
38 [...]
39
40 )))))))))->isEmpty()
41
42 ))->size()
```

Listing 5.6: Injection threat scenario formalized using OCL

Threat Analysis Results

Threat categories	**Detected threats**	**TP**	**FP**	**FN**
Man-In-The-Middle	2	2	0	0
Tampering	2	2	0	0
Denial of Service	3	2	1	0
Injection	3	2	1	0
Total	10	8	2	0

Table 5.9: Number of threats per scenario

After checking OCL constraints over the working example, we obtain the results presented in Table 5.9.

Evaluation Metrics Results

From threat analysis results, we compute *Precision*, *Recall* and *F-Measure* metrics. The results are shown in Table 5.10. The actual version of the threat scenarios formalized with OCL has a Precision, Recall and F-Measure rates of 80%, 100% and 85,88% respectively. The precision rate, that measures soundness, indicates that 20% are FN. In order to increase the precision we must consider port kinds (internal or external). Indeed, server and database communicating ports have been considered internal in the working example in Figure 2.11 during Microsoft's threat modeling.

5.4.5 Iteration 4

In this iteration, we add the concept of port kinds. Figure 5.3 shows ComponentUML with the *PortKind* enumeration with two literals *external* (public ports) and *internal* (private ports).

Metrics	**Values**(%)
Precision	80
Recall	100
F-Measure	85,88

Table 5.10: Evaluation metrics results

Threat scenarios formalization

We consider trust level of components and obtain a new version of the OCL constraints. In Listings 5.7 and 5.8 are given extracts of these OCL constraints. Note that a full listing of the OCL constraints can be found in Appendix B.

```
1 Context Application inv DenialofService_v2
2
3 self.components->select(c1 |
4 /*Publicly accessible port*/
5 c1.ports->exists(public_port |
6 (public_port.portkind = PortKind::external)
7 and
8 /* Firewall, authentication and authorization mechanisms exist
9 [...]
10 /*Security mechanisms are connected to component c1 and message flow is correct*/
11 [...]
12 )->size()
```

Listing 5.7: Denial of Service (DoS) threat scenario formalized using OCL

```
1 Context Application inv Injection_v2
2
3 self.components->select(c1 |
4 /*Publicly accessible port*/
5 c1.ports->exists(public_port |
6 (public_port.portkind = PortKind::external)
7 and
8 /* Firewall, authentication and authorization mechanisms exist
9 [...]
10 /*Security mechanisms are connected to component c1 and message flow is correct*/
11 [...]
12 )->size()
```

Listing 5.8: Injection threat scenario formalized using OCL

Threat Analysis Results

After checking OCL constraints over the working example, we obtain the results in Table 5.11.

Threat categories	Detected threats	TP	FP	FN
Man-In-The-Middle	2	2	0	0
Tampering	2	2	0	0
Denial of Service	2	2	0	0
Injection	2	2	0	0
Total	8	8	0	0

Table 5.11: Number of threats per scenario

Evaluation Metrics Results

From threat analysis, we compute *Precision*, *Recall* and *F-Measure* metrics. The results are shown in Table 5.12. The fourth version of the threat scenarios formalized with OCL has a Precision, Recall and F-Measure rates of 100%. This means that these formalizations allow the detection of exactly the same threats of the web application in Figure 2.11 using Microsoft's threat modeling.

Metrics	**Values**(%)
Precision	100
Recall	100
F-Measure	100

Table 5.12: Evaluation metrics results

5.5 Tool Support

We have rapidly described a global process for automating threat analysis based on formalized threat scenarios. Indeed this approach requires first an improved architecture description formalism. Second, the relevant threat scenarios have to be formalized using OCL.

In order to support our approach, tools must fulfill requirements in section 4.6.1 and the following key ones:

- Allows the creation of OCL constraints and the specification of OCL invariants (step 0).
- Allows the validation of OCL constraints over a UML model (step 1).
- Allows report generation.

In our case, we have chosen the following support tools :

- OCL constraint specification: included in Papyrus.
- OCL verification module : Papyrus DSML Validation Feature[1].

5.6 Illustration

In this section, we use formalized threat scenarios in *Iteration 4* over the Microsoft's web application architecture secured with the SSL pattern.

5.6.1 Software architecture and platform

We recall that the software architecture of the application presented in Figure 4.24 was obtained with the integration process illustrated in section 4.5. This architecture is deployed on a platform described in Figure 5.2. Table 5.13 describes this deployment.

Software Components	**Hardware Nodes**
client (browser)	client
encryptor	client
decryptor	client
signer	client
verifier	client
client_ProtocolController	client
key_Exchange	client
authenticator	client
website	web server
encryptor_1	web server
decryptor_1	web server
signer_1	web server
verifier_1	web server
server_ProtocolController	web server
key_Exchange_1	web server
authenticator_1	web server
database	Database server

Table 5.13: Deployment of the software component on the hardware nodes

[1]https://wiki.eclipse.org/Papyrus/UserGuide/Profile_Constraints

5.6.2 Selection of Constraints

The selection of OCL constraints is allowed by a pop-up that lists the available constraints. We select the four threat scenarios formalized using OCL as shown in Figure 5.4.

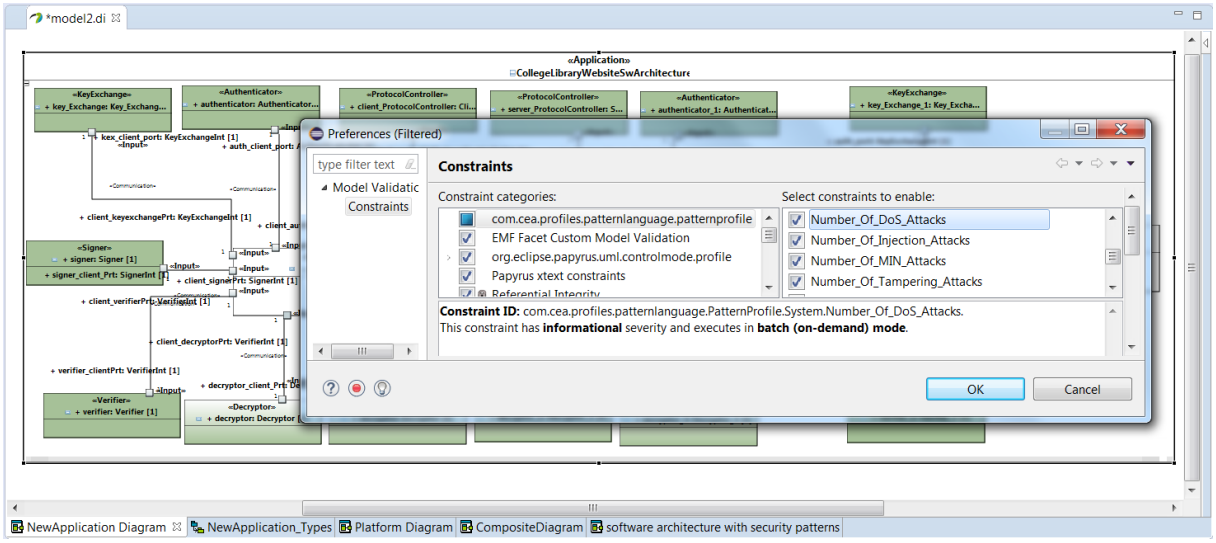


Figure 5.4: Selection of Constraints to be enabled during checking

5.6.3 Results

The results of threat analysis are presented in Figure 5.5. The results show that there are no threats relevant to Man-In-The-Middle (MITM) or Tampering threat scenarios and 2 threats relevant to Denial of Service (DoS) and Injection threat scenarios. The integration of the SSL pattern provides confidentiality, authenticity and integrity properties between client and college server. Thus, this pattern stops Man-In-The-Middle (MITM) and Tampering threats. However, there are two residual threats relevant to Denial of Service (DoS) and Injection threat scenarios. This is due to a lack of Firewall and Authorization mechanisms for client and server components.

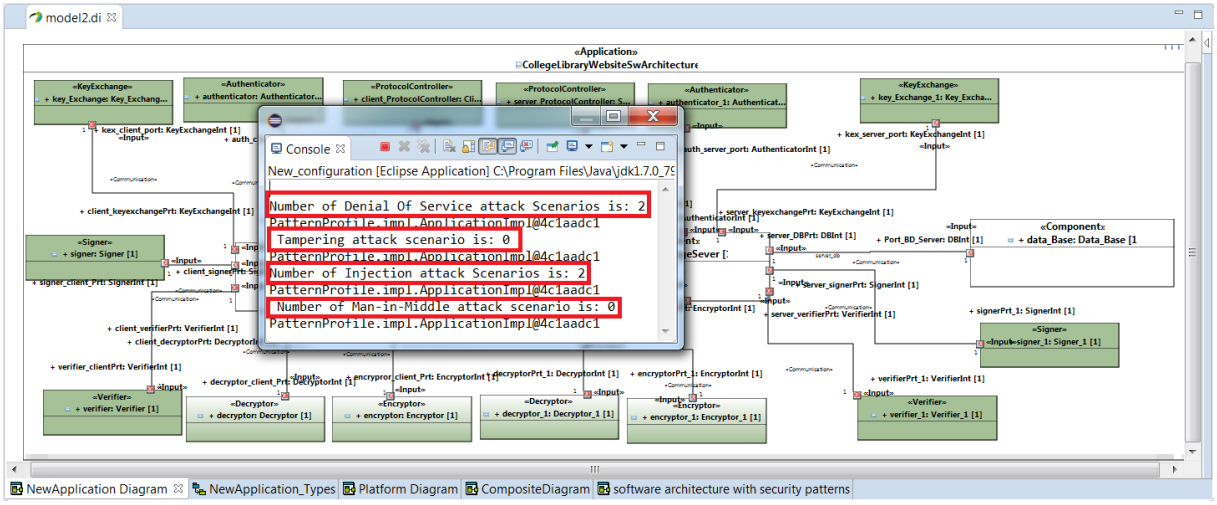


Figure 5.5: Threat analysis results

5.7 Conclusion

In this chapter, we have answered a part of **RG1**, **RG3** and **RG4**. We have proposed a threat scenario-based approach for analyzing a software architecture. We have used OCL to formalize Denial of Service (DoS), Tampering, Injection and Man-In-The-Middle (MITM) threat scenarios. We also explain the formalization process through four iterations. For each iteration, we evaluate the soundness and completeness of the threat scenarios formalization.

The contributions of this work are twofold. First the approach enables a detailed exploration of the software architecture. The formalized threat scenarios allow not only the verification of the existence of security mechanisms but also the verification of their correct usage. Second, the approach fosters reuse and extensibility of threats scenarios thanks to a rigorous formalization. In fact, we have provided a formalization process based on iterations and evaluation metrics. This formalization process may be reused in order to formalize other threat scenarios. In addition, the formalized threat scenarios are application independent and are stored together with ComponentUML. As a future work, we intend to use common Component-Based Modeling Languages e.g., Unified Component Model (UCM) [131]. In addition, we will simplify the OCL rules with a library of helpers that will allow security experts to contribute to the formalization to threat scenarios with less effort.

In the next chapter, we discuss the analysis of software architecture candidates resulting from different System of Patterns configurations with regards to non-functional

requirements. We focus more on real-time aspects. The aim is to analyze these candidates (with the same security properties but can having different impact on performance) with regards to real-time requirements.

Chapter 6

Real-Time Analysis of Software Architectures

Contents

6.1 Introduction

ICTs generally involve multi-concern objectives. In this context, software systems must satisfy a number of requirements (real-time, physical, energy efficiency and others). Security concerns have an impact on other concerns. Therefore, architects must apply trade-offs to satisfy non-functional requirements and security requirements.

In this chapter, we focus on the analysis of software architecture solution candidates. It is important to check if architecture candidates meet other non-functional requirements (in our case real-time). Actually, adding security mechanisms may impact response time and/or resource consumption. It is thus important to be able to evaluate that such requirements are still satisfied after application of patterns. We extend previous works of the team [121, 62]. In [121], a MARTE-based framework for real-time schedulability analysis for early design stages was presented. In [62], an approach to support Security, De-

pendability and Resource Trade-offs using Pattern-based Development and Model-driven Engineering was presented. Here we go one step further. We propose a process and its tool support for the analysis of security software architecture candidate solutions against real-time requirements (to answer a part of: **RG1**, **RG3** and **RG4**). The aim here is to evaluate the impact of security patterns on real-time performance, as visualized in *activity A3.3* of Figure 3.1.

Previous works have focused on security and real-time requirements separately: dependability and security modeling and analysis [23][24] and real time requirements [109][69]. A survey of dependability modeling and analysis frameworks with UML can be found in [24]. The approaches focus on software systems Reliability, Availability, Maintenance and Safety (RAMS). Many methods support early life cycle phases (from requirements to design). The survey indicates that the tool support needs more automation and has to return analysis results directly to the designer environment. In [23], the authors have extended MARTE with a Dependability Analysis and Modeling (DAM) UML profile and applied it to an intrusion-tolerant message service case study. The results showed a need of dependability annotations. In [109], the authors presented a staged approach to optimize the deployment in the context of real-time distributed systems. Harbour et al. presented in [68], MAST (Modeling and Analysis Suite for Real-Time Applications). It is a tool suite that allows the modeling and schedulability analysis of timing behavior of real-time systems. The tool suite was extended in [69] to enable an automatic schedulability analysis of a distributed application using switched networks and clock synchronization mechanisms.

The remaining sections are organized as follows: Section 6.3 presents the main steps of real-time analysis approach for software architecture candidates. Section 6.4 presents the approach with the used modeling languages and is illustrated via Microsoft's web application working example. For the illustration, we use the following techniques: (1) Fixed offset-based scheduling with Rate-Monotonic Analysis (RMA) [153] and (2) Deterministic (static) Task Partitioning strategy. In section 6.5, we give an overview of the tool support requirements and describe the used one. In section 6.6, we conclude and sum up the contributions.

6.2 Related Work

The analysis of the overhead of security with regards to quality attributes is not new. However the analysis of the integrated security solution with regards to the overall system architecture quality and performance is a fresh topic. The contribution of this work is

a step towards this goal. It presents a model-based method for evaluating security pattern-based architectures for decision purposes. The analysis is done in the context of real-time analysis. We distinguish two categories of related works: (1) approaches that evaluate security solutions with regards to other non-functional requirements; (2) approaches dealing with general trade-off analysis between several non-functional requirements.

6.2.1 Analysis of software architecture solutions with regards to real-time Requirements

Concerning security solution evaluation, similarly to our approach, in [33] the authors propose a framework to evaluate the impact of using security mechanisms. The approache uses languages such as UML [130] and MARTE [128] for modeling the architecture and security mechanism components. They also use predefined security components for security solutions. The difference is that in our approach, security concerns are handled through the application of security patterns rather than direct use of predefined security components. Woodsite et al. [162] focused on the analysis of the performance effects of security solutions modeled as UML non-functional aspects. They used SPT UML profile for annotating a UML design with schedulability, time and performance data. The resulting model and the security aspects were transformed separately and composed into one model which was then analyzed. In [104], the authors present an automatic approach for improving quality attributes (performance, cost, configuration options) of a system architecture. The degrees of freedom depend on design options (e.g., component allocations, processing rates). The difference with our approach is that it acts only on the initial system architecture without refinements. However, this approach can be used to optimize the refined architectures obtained after pattern integration.

6.2.2 Architecture Decision and Trade-off Analysis

Other works focused on large scale architecture optimization, decision and trade-off analysis. The Architecture Trade-off Analysis Method (ATAM) [87] and the Software Architecture Analysis Method (SAAM) [86] are well-known methods for performing assessment of quality attributes for deciding between several architecture design alternatives. The SVDT [76] approach is security-oriented and uses UMLsec [85] for modeling security solutions and modules to check their validity with regards to other Non-Functional Requirements (NFPs). The validated solutions are evaluated using security solution trade-off analysis. The main difference with our approach is that we consider a higher level model-

ing and solutions are expressed in terms of patterns. In the automotive domain, a multi-objective automatic optimization approach based on EAST-ADL modeling is proposed [159]. It supports the evaluation of alternative architectures according to dependability, timing performance, cost etc. A similar work in [98] presented a method for the search of optimal architecture design according to multi-objectives such as cost, performance and reliability based on SysML modeling. In [46], the authors identified limitations in UML language to support architecture decision according to non-functional attributes. They propose a framework based on parametric analysis specification that aims at evaluating design decisions.

6.2.3 Positioning

The difference between our work and the aforementioned works is as follows:

- *Security Solution Analysis with regards to NFPs.* Our approach is part of this category as it focuses on the analysis of architecture candidates with regards to real-time requirements. The approach makes extensive use of UML and MARTE. In [33], it is not clear how MARTE is used to give feedback of the results to the user. In addition, our input architecture models represent a software architecture with applied security patterns which is not the case in [33] where an analysis model is used with annotations referring to some security mechanisms. We can benefit from Martens et al. in [104] to improve a chosen security-pattern-based architecture according to quality attributes (performance, cost, configuration options) and in that sense this work completes our approach.

- *Architecture Decision and Trade-off Analysis.* These approaches allow trade-off analysis between several architectures or optimize quality attributes of certain architecture. In the two cases, these approaches are complementary to our approach.

6.3 Analyzing software architectures with regards to real-time requiments

In this section we present an overview of the proposed methodology in Figure 6.1. The aim here is the analysis of software architecture candidates with regards to real-time requirements. It is important to recall that these candidates may be obtained because there can be different System of Patterns configurations (see Figure 4.16). This process

relies on two main artifacts: (1) a software architecture candidate and (2) a platform model with deployment. Hence the process is performed for each candidate.

6.3.1 Methodology description

As described in Figure 6.1, the software architecture candidate and the platform models are used to model end-to-end flows (step 1). Each end-to-end flow contains a sequence of functions connected with each other and triggered with an event. The functions represent operations and the end-to-end flow represents a message sequence in the software architecture. For each end-to-end flow, timing parameters are added (step 2): end-to-end flow deadlines, worst case execution times (WCET) and worst case transmission times (WCTT), activation event patterns (periodic, aperiodic, sporadic). A task model is generated based on the end-to-end flows architecture, the platform specification and the deployment (step 3) and submitted to schedulability (step 4). The task model is schedulable if all tasks are schedulable.

6.3.2 End-to-end Flows Modeling (Step 1)

The modeling process starts by eliciting all end-to-end flows in the software architecture according to existing messages. An end-to-end flow consists of a set of communicating functions from one end to the other. For example, in Figure 6.1, the messages in the software architecture are used to model *End-To-End Flow 1* which consists of functions: *Login*, *Encrypt*, *Decrypt* and *OpenSession*.

6.3.3 Timing parameters (Step 2)

End-to-end flows are then annotated with timing parameters: (1) end-to-end flows deadlines, (2) WCET and WCTT for functions and communications (3) activation event patterns (periodic, aperiodic, sporadic) and activation periods. In addition, The platform model is annotated with execution hosts and channels maximum capacities.

6.3.4 Task Model Generation (Step 3)

In this step, the task model is generated from the end-to-end flows and the deployment on the platform according to task partitioning and allocation techniques. A survey was made in [88] and listed different strategies and techniques. We assume that tasks are periodic. In the illustration, we use a deterministic (static) task partitioning and allocation technique.

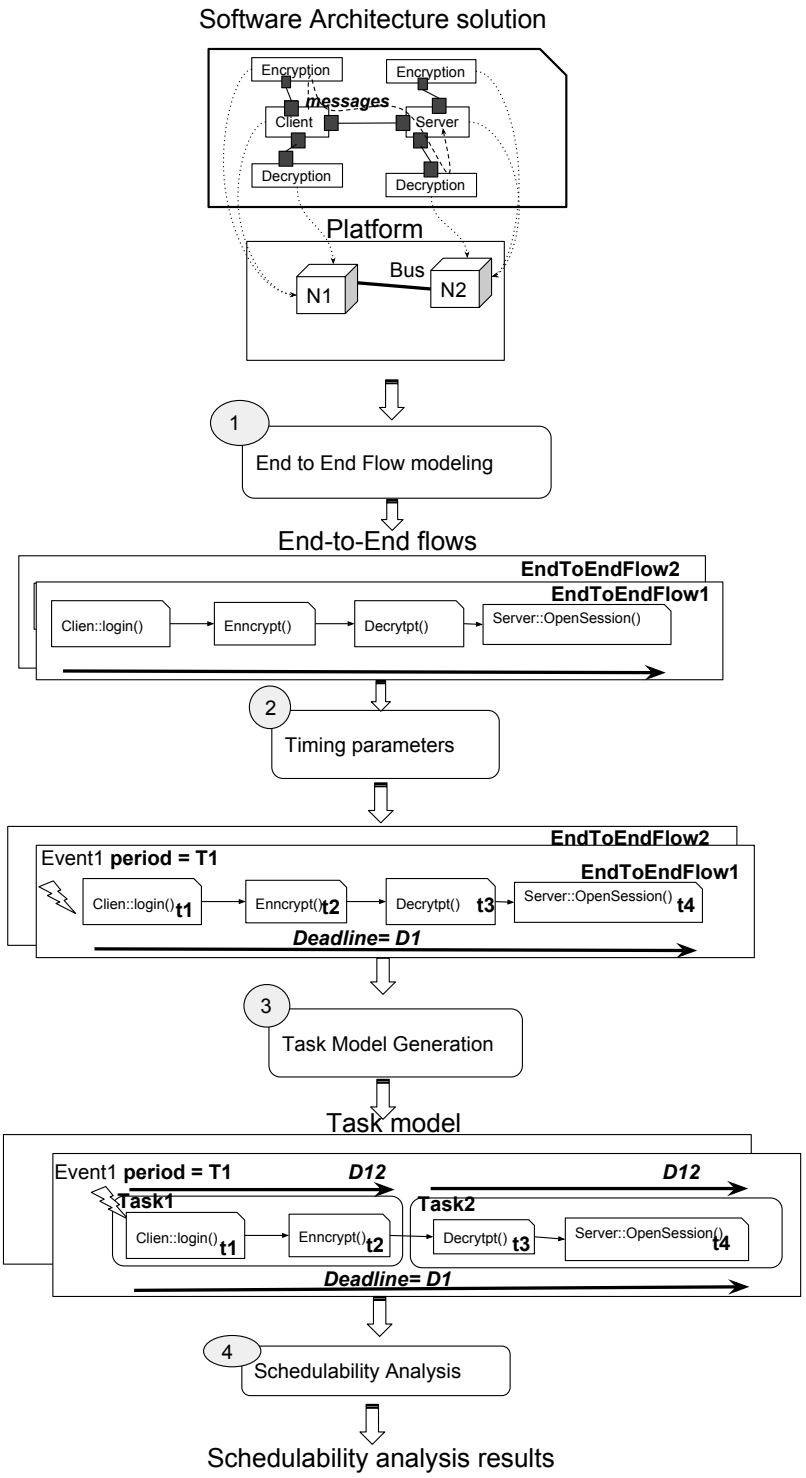


Figure 6.1: Real-time analysis of software architectures process

6.3.5 Schedulability Analysis (Step 4)

Once the task model is generated, each task must be performed before its deadline according to a specified scheduling policy. We assume that the system is distributed i.e., the platform consists of more than one execution node. Hence the scheduling policy must be adapted for distributed systems. Davis et al. [36] have made a survey of scheduling policies for real-time multi-processors. In this illustration, we use fixed priority offset-based scheduling with Rate-Monotonic Analysis (RMA) [153].

Let P be a platform consisting of nodes $\{N_1,N_2,\ldots,N_a\}$ connected through channels $\{B_1,B_2,\ldots,B_b\}$. We denote F as a set of functions $\{f_1,f_2,\ldots,f_c\}$ and S the set of communications $\{s_1,s_2,\ldots,s_d\}$. Each function and communication has a WCET ($c_{(f_i)}$) and a WCTT ($c_{(s_i)}$) respectively. Both nodes and channels have a maximal capacity $c_{max}(Ni)$ and $c_{max}(Bi)$ that must not be exceeded. In the context of multiple processors, each execution node N_i runs an independent real-time operating system and executes a set of tasks:

$$T_i = \{t_{(i,1)}, t_{(i,2)}, \ldots, t_{(i,ni)}\} \tag{6.1}$$

Each task $t_{(i,j)}$ consists of a subset of functions from F and is characterized with an activation period $P_{(i,j)}$, a total WCET $c_{(i,j)}$ (computed as the sum of WCETs of all allocated functions) and a deadline $D_{(i,j)}$. Scheduling analyzes generally consists of two steps:

1. **Processor utilization:** is used to compute nodes utilizations. All node and channel utilizations U_{Ni} and U_{Bi} must be less than $c_{max}(Ni)$ and $c_{max}(Bi)$ respectively. In this case we move to step 2. If not, the node is overloaded and response time analysis is not performed.

$$U_{Ni} = \sum_{t_{(i,j)} \in T_i} \frac{c_{(i,j)}}{P_{(i,j)}} \tag{6.2}$$

$$U_{Bi} = \sum_{s_i \in Bi} \frac{c_{(s_i)}}{P_{(s_i)}} \tag{6.3}$$

$$U_{Ni} \leq c_{max}(Ni) \tag{6.4}$$

$$U_{Bi} \leq c_{max}(Bi) \tag{6.5}$$

In case of RMA [97], c_{max} is equal to 1.

2. **Worst case response-time computation:** concerns the computation of the Worst Case Response Time (WCET) $R_{(t_{(i,j)})}$ of every task. In case of fixed-priority offset-based scheduling with RMA [153], each task is assigned a fixed priority. The shortest the task, the higher the priority.

   The task model is schedulable if for each task $t_{(i,j)}$:

$$R_{(t_{(i,j)})} \leq D_{(i,j)}. \tag{6.6}$$

6.4 Model-Based Real-Time Analysis

In this section we describe an MDE framework to support real-time analysis. In section 6.4.1, we describe the modeling principle based on MARTE in order to support real time analysis process presented in section 6.3. In the next sections, we illustrate the steps of the process via Microsoft's web application working example. For illustration purposes, we consider one software architecture candidate obtained with the integration of the SSL pattern illustrated in section 4.5. The input software architecture and platform are shown in Figure 4.24 and Figure 5.2.

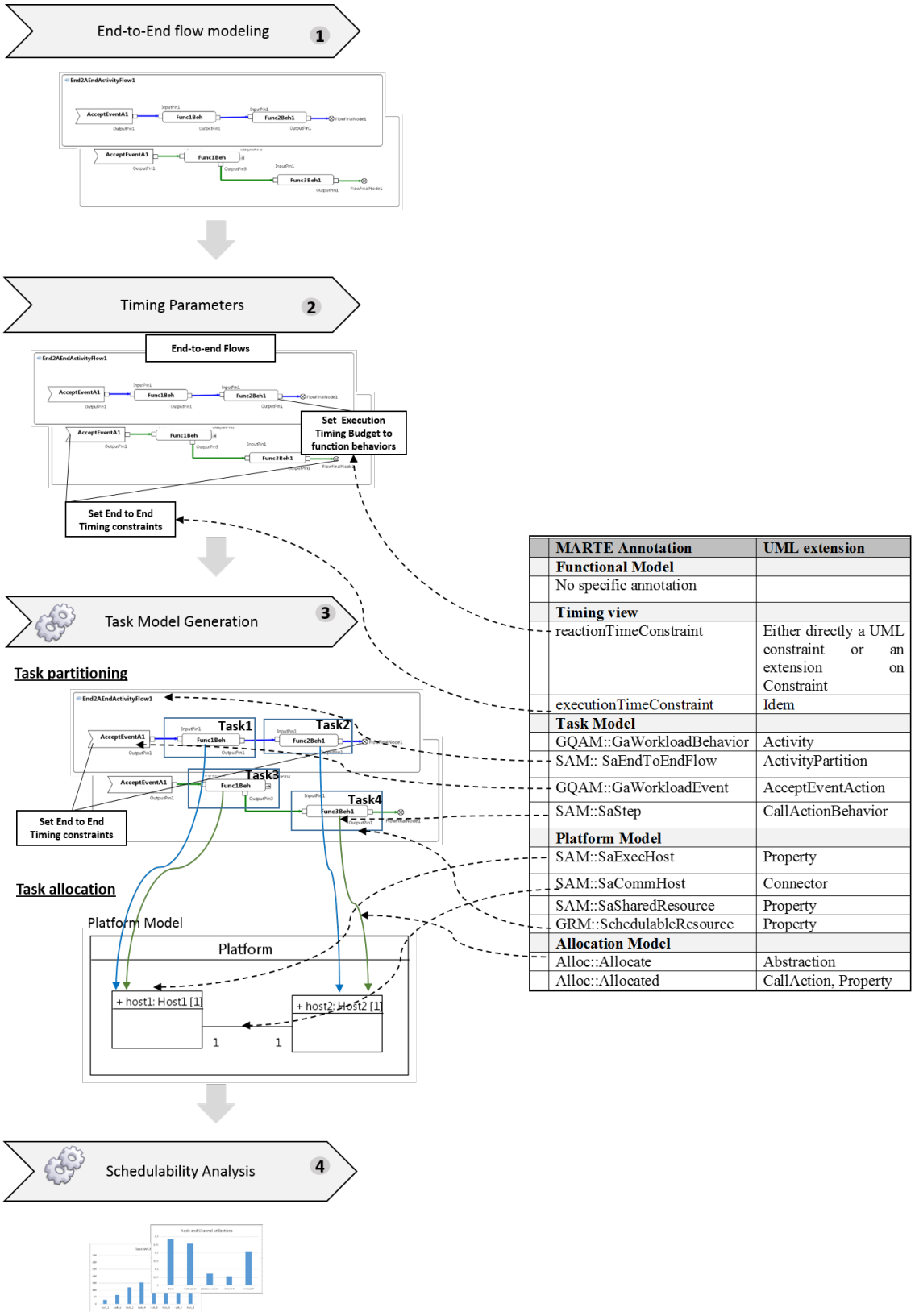
We also use the following techniques:

- Scheduling policy : *Fixed offset-based scheduling with Rate-Monotonic Analysis (RMA)* [153].
- Task partitioning and allocation : Deterministic (static) Task Partitioning strategy.

6.4.1 Model-based analysis with MARTE

The process presented in Figure 6.2 is supported with MARTE (Modeling and Analysis of Real-Time and Embedded systems) [128] as shown in Figure 6.1. MARTE is a UML profile domain-specific modeling language for model-based design and analysis of real-time embedded software of CPSs. MARTE [1] is an OMG standard that was designed to supplement UML with capabilities relevant to real-time systems that are missing or poorly supported in UML. In our context, we use a subset of MARTE in order to model: (1) end-to-end flows, (2) timing parameters and (3) task models. Table 6.1 shows the used stereotypes .

[1] http://www.omg.org/spec/MARTE/



	MARTE Annotation	UML extension
	Functional Model	
	No specific annotation	
	Timing view	
	reactionTimeConstraint	Either directly a UML constraint or an extension on Constraint
	executionTimeConstraint	Idem
	Task Model	
	GQAM::GaWorkloadBehavior	Activity
	SAM:: SaEndToEndFlow	ActivityPartition
	GQAM::GaWorkloadEvent	AcceptEventAction
	SAM::SaStep	CallActionBehavior
	Platform Model	
	SAM::SaExecHost	Property
	SAM::SaCommHost	Connector
	SAM::SaSharedResource	Property
	GRM::SchedulableResource	Property
	Allocation Model	
	Alloc::Allocate	Abstraction
	Alloc::Allocated	CallAction, Property

Figure 6.2: Model-Based Real-Time Analysis with MARTE

MARTE Stereotype	UML extension
End-to-end flows and Task Model stereotypes	
GQAM::GaWorkloadBehavior	Activity
GQAM::GaWorkloadEvent	AcceptEventAction
SAM:: SaEndToEndFlow	ActivityPartition
SAM::SaStep	CallActionBehavior
Alloc::Allocate	Abstraction
Alloc::Allocated	CallAction, Property
Platform Stereotypes	
GQAM::GaPlatformResources	Class
SAM::SaExecHost	Property
SAM ::SaCommHost	Connector
SAM::SaSharedResource	Property
GRM::SchedulableResource	Property
Allocation Model	
Alloc::Allocate	Abstraction
Allocated	CallAction, Property

Table 6.1: MARTE stereotypes and extensions

6.4.2 End-to-end Flows Modeling (Step 1)

The end-to-end flows and the platform are modeled using a subset of MARTE based on the software architecture model. For illustration purposes, we only consider one software architecture candidate presented in Figure 4.24. The platform is described in Figure 5.2 and Table 5.13 describes the deployment of software components on the platform nodes. We also consider one use case called 'User Logging Securely'. Figure 6.3 describes sequence diagram of this use case. This sequence diagram is mandatory to model the end-to-end flow used for real-time analysis. As shown in Figure 6.4, the modeled end-to-end flow consists of an *SaEndtoEndFlow* that consists of a set of *SaSteps* connected with *SaCommSteps* and triggered with a *GaWorkLoadEvent*.

6.4.3 Timing Parameters (Step 2)

End-to-end flows are then annotated with timing parameters: (1) end-to-end flows deadlines (*reactionTimeConstraint*), (2) WCET and WCTT for functions and communications (*executionTimeConstraint*) (3) activation event pattern (periodic, aperiodic, sporadic) and its period (*ArrivalPattern*). In addition, the platform model is annotated with execution hosts (*SaExecHost*) and channels (*SaExecComm*) maximum capacity.

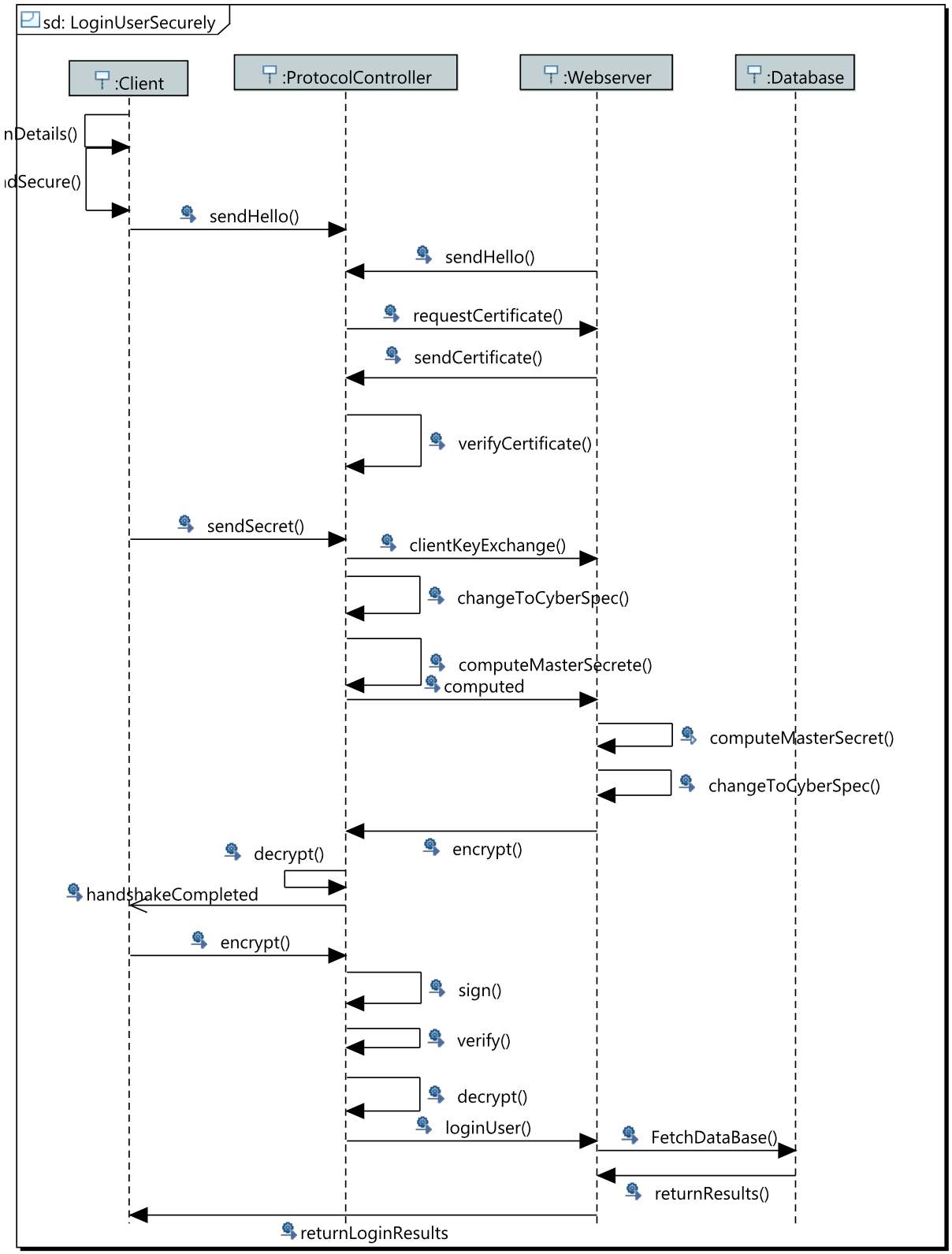


Figure 6.3: Sequence diagram for the use case 'User Logging Securely'

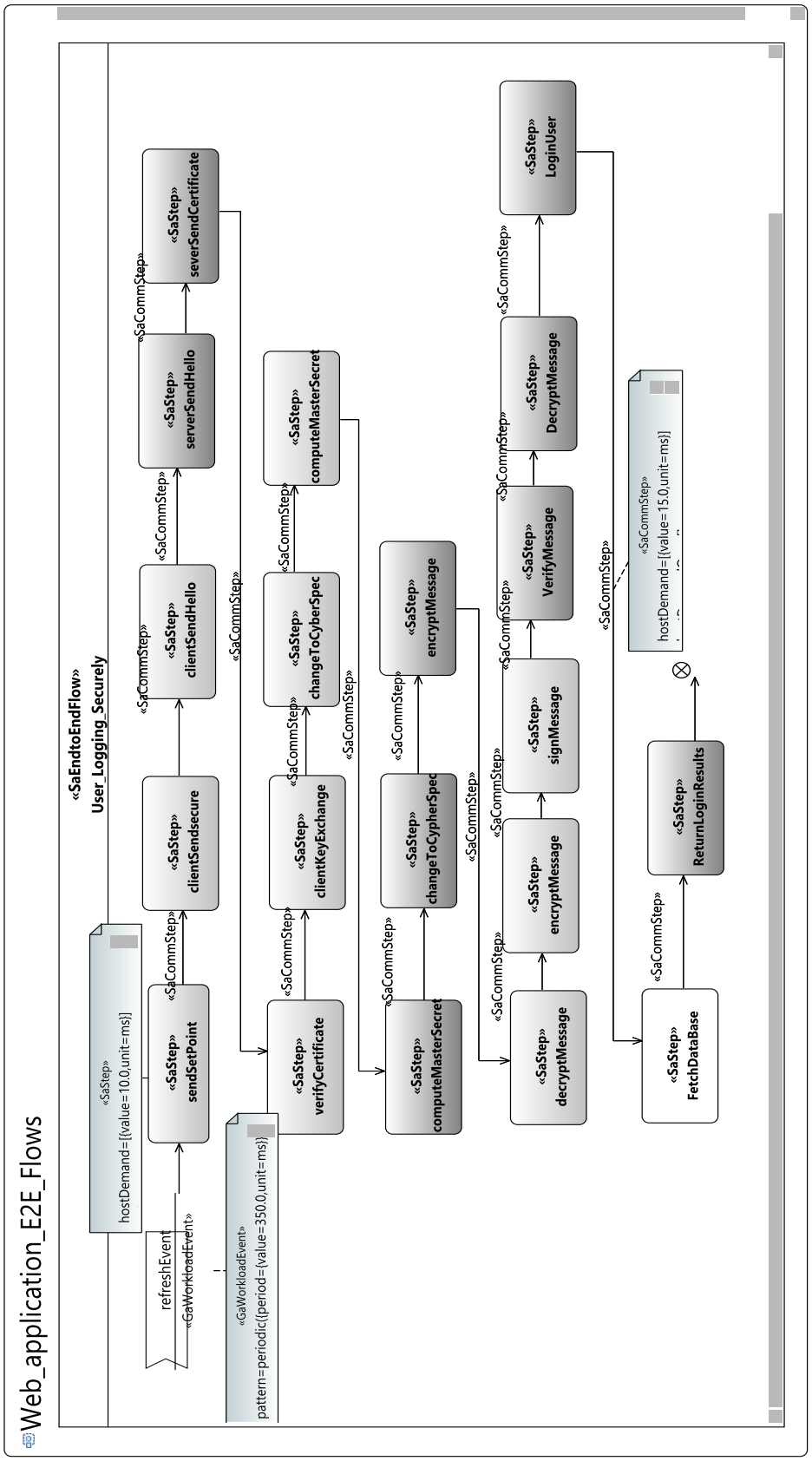


Figure 6.4: End-to-end flow for the use case 'User Logging Securely'

Illustration

SaStep	WCET (ms)
submitLoginDetails	10
sendSecure	10
sendHello	10
sendCerificate	10
verifyCertificate	10
clientKeyExchange	10
changeToCyberSpec	10
computeMasterSecret	10
changeToCyberSpec	10
encryptFinishedMessage	10
decryptFinishedMessage	10
encryptMessage	10
signMessage	10
verifyMessage	10
decryptMessage	10
loginUser	10
FetchDataBase	25
returnLoginResults	10

Table 6.2: SaSteps Execution times (WCET)

SaCommStep	WCTT (ms)
client to server	10
server to database	15

Table 6.3: SaCommSteps Execution times (WCTT)

End-to-end flow 'Logging User Securely' in Figure 6.4 is annotated with:

- Event periods for each GaWorkloadEvent. The end-to-end flow is triggered with a periodic event of 300 ms.
- WCET and WCTT for each *SaStep* and *SaCommStep* respectively. WCETs and WCTTs are weighted values and used only for illustration purposes. Table 6.2 and Table 6.3 show the WCETs and WCTTs of each SaStep annotated to the end-to-end flow.
- End-to-end flow deadline of 300 ms.

The Platform maximum capacity of the nodes and communications is assumed to be 100%.

6.4.4 Task Model Generation (Step 3)

Task model generation is performed in two steps. Each step is performed according to specific techniques:

1. *Task Partitioning.* The task model structure is described using activity diagrams. It can be directly obtained from the end-to-end flows and the deployment of functions onto execution nodes and communications into channels. Each task is represented with a parent *SaStep* nesting a set of *SaSteps* and *SaCommSteps*.

2. *Task Allocation.* Finally an allocation model is described in a composite diagram that shows the allocation (*Alloc::Allocate*) from tasks to execution nodes and from communications to channels.

Illustration

The task model is generated according to a static task partitioning and allocation technique. The task model consists of eight tasks as represented in Figure 6.5. Each task consists of a set of *SaSteps* and is allocated in an execution node. The task allocation is described in Figure 6.6.

6.4.5 Schedulability Analysis (Step 4)

The task model is submitted to schedulability analysis. Tasks of the same node are assigned a priority according to RMA [97]. Node and channel utilizations are shown in Figure 6.7. The utilizations are less less than 1 for each node. Figure 6.8 gives the WCRT of the tasks. All tasks respect their deadlines.

6.4.6 Discussion

The analysis shows that the software architecture with overhead induced by the SSL pattern respects real-time constraints within the specified schedulability policy. In addition, in case of multiple software architecture candidates produced by integration of different System of Patterns configurations, this work can be beneficial. It can enable the architect

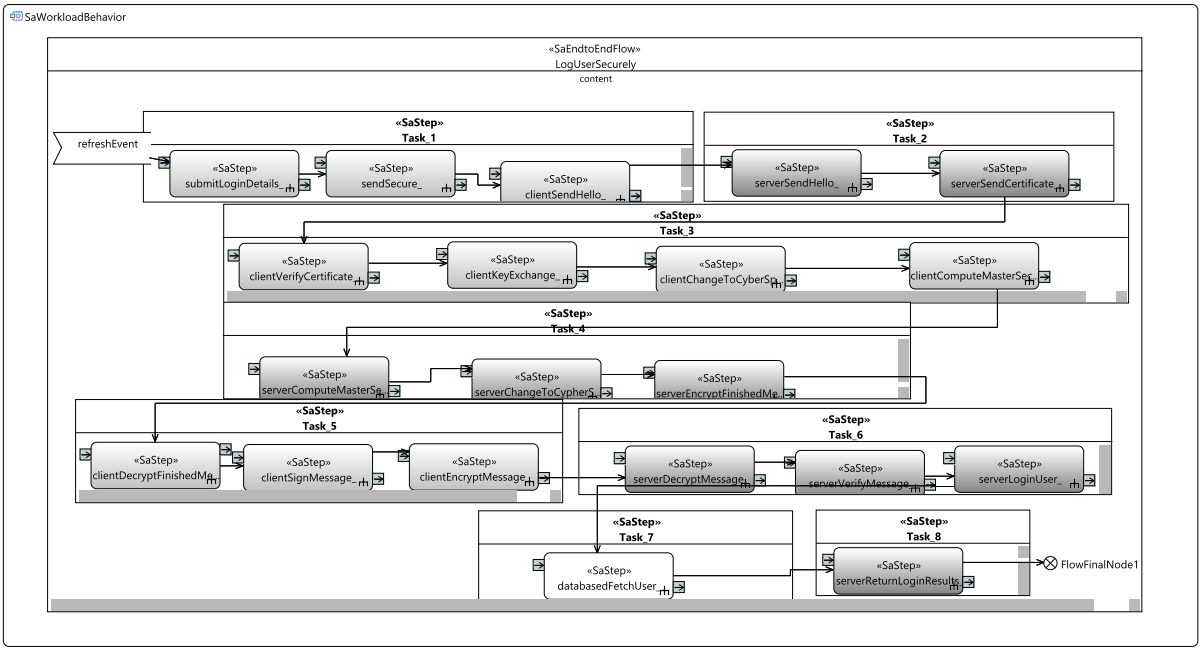


Figure 6.5: Tasks partitioning

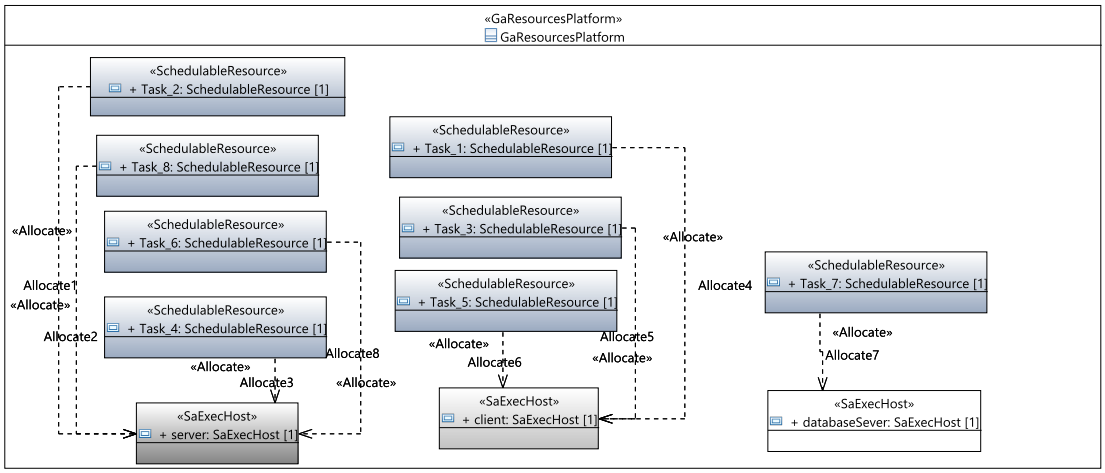


Figure 6.6: Tasks allocation

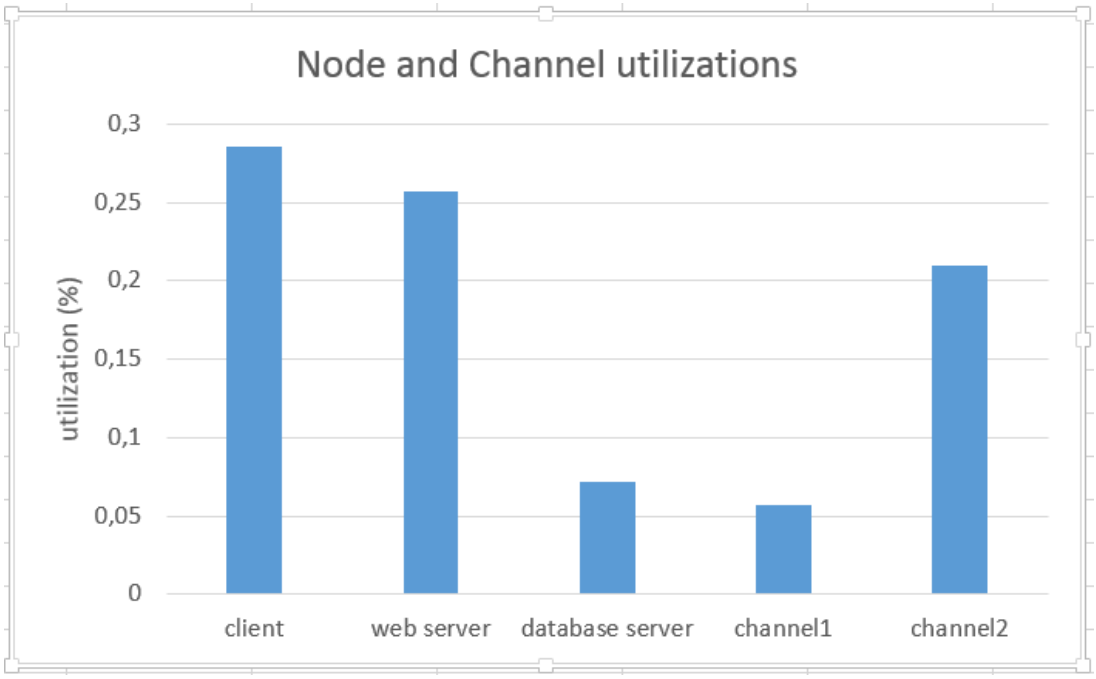


Figure 6.7: Node and channel utilizations

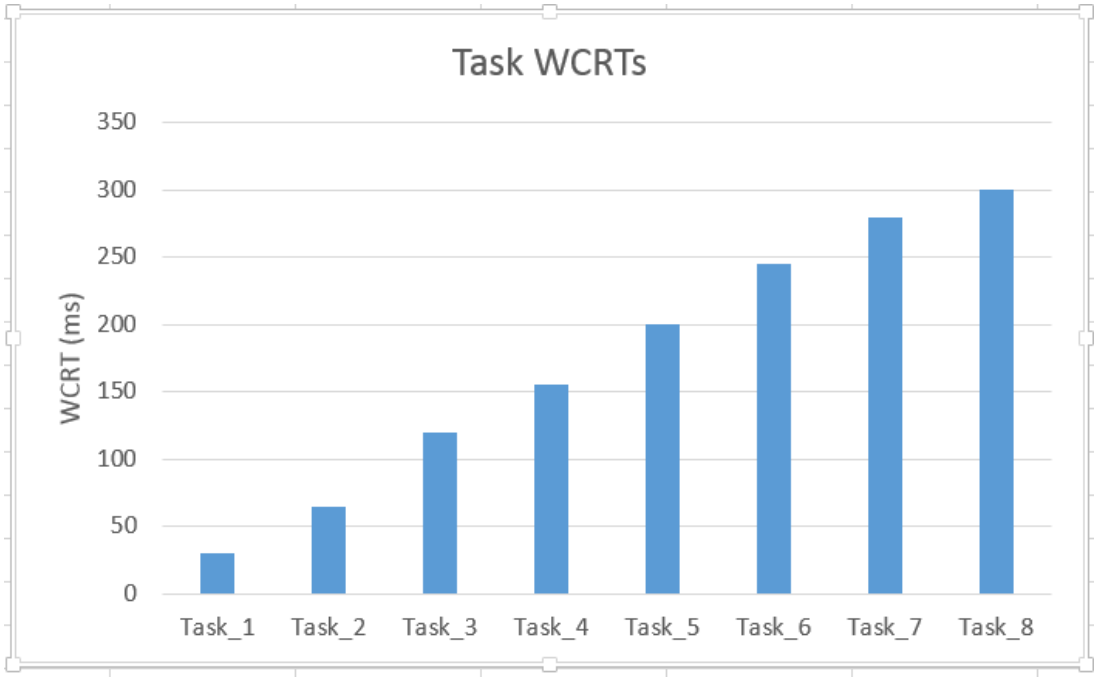


Figure 6.8: Task WCRTs

to have early evaluation results and decide between the architecture alternatives according to specified criteria. In the worst case scenario and if the evaluation results of all the architecture alternatives are not satisfactory, the architecture can be rethought.

6.5 Tool Support

Appropriate tools for supporting the approach must fulfill the following key requirements:

- Enable the creation of the UML models used to describe system and software architecture.
- Allow the creation of a custom UML profile.
- Support the implementation of a repository to store pattern models and the related model libraries for classification and relationships.
- Enable the creation of visualizations of the repository to facilitate its access.
- Support the access to the repository. Create views on the repository according to its APIs, its organization and the needs of the targeted system engineering process.
- Enable transformations of the pattern models from the repository format into the target-modeling environment (Papyrus pattern format).
- Enable the creation of System of Patterns in the target-modeling environment.
- Enable the creation of pattern configuration models in the target-modeling environment.
- Enable the integration of application models and models imported from the repository.
- Support the calculation of resource consumption and real-time scheduling.
- Provide the ability to create customized reports by querying the resulting models.

Amongst the existing alternatives, we have chosen Papyrus[2] to create UML diagrams for the model of the application. SEMCOMDT [3](SEMCO Model Development Tools, IRIT's editor and platform plugins) is used to support pattern repository. The generation

[2]https://eclipse.org/papyrus/
[3]http://www.semcomdt.org

of System of Patterns configurations from a System of Patterns has been described in [62]. The MARTE UML profile is already integrated into Papyrus. Real-time analysis have been performed using a tool called "Qompass Architect" [92] integrated in the Papyrus environment. It is a model-based tool developed in CEA LIST for QoS assessment and optimization of real-time architectures. Qompass Architect explores non-functional properties of real-time architectures to finally synthesize an optimized architecture. Note that other tools performing schedulability analysis can be used such as cheddar[146].

6.6 Conclusion

In this chapter, we answered a part of **RG1**, **RG3** and **RG4**. We have proposed a process and its tool support for the analysis of software architecture candidates of the application against real-time requirements.

The main benefits rely on providing a tooling support to allow early evaluation of software architecture candidates. The results obtained help the architect to evaluate the overhead of security patterns on the software architecture to reinforce security and to compare possible alternative solutions. The methodology relies on UML and MARTE for the modeling to perform architectural analysis for timing concerns. As a future work, we want to address other concerns such as the interplay of safety and security.

In the next chapter, we assess the feasibility of the contributions (from **RG1** to **RG4**) through the modeling and analysis of a SCADA system case study (to answer **RG5**).

Chapter 7

Assessment of the contributions

Contents

7.1 Introduction

This chapter assesses the feasibility of the contributions of our work (from **RG1** to **RG4**) through the modeling and analysis of a SCADA (Supervisory Control And Data Acquisition) system (to answer **RG5**). We discuss the assessment of the results for each contribution. The selection of an application for the proposed approach has to be considered as a very important decision for our work. It has a direct impact on the selection of patterns, and thus on the whole assessment. The main characteristic considered to select the SCADA system application was the fact that these system are target to many attacks (e.g., March 2016 event [6]). In fact, the key issue nowadays is SCADA security and governments all over the world are worried about the security of SCADA systems that run over critical infrastructures. In addition, SCADA system applications are different from classical ICTs and have strong security demands.

The remainder of this chapter is organized as follows. In section 7.2, we describe the SCADA system case study including the platform and the software architecture. We also give an overview of SCADA system security issues and their general impact on performance. Then, we model the SCADA application using our tool chain and describe the obtained results. section 7.3, discusses the feasibility of the approach by assessing the

results obtained for each contribution. Finally, we sum up the assessment and conclude in section 7.4.

7.2 SCADA case study

7.2.1 Description

SCADA stands for Supervisory Control and Data Acquisition systems. As its name indicates SCADA systems are meant to continuously control, monitor processes and acquire field information. The applications of SCADA systems are various. They are used in most industrial processes (e.g. power generation and distribution in the context of smart grids, chemistry, food industry, etc.).

In the following sections we use the case study displayed in Figure 7.1. It shows a typical SCADA architecture used in the context of smart grids [161]. In this context, the controlled process is power distribution. It consists of a control center and a number of field devices connected by a communication infra-structure. Field devices can be Programmable Logic Controllers (PLC), sensors and actuators.

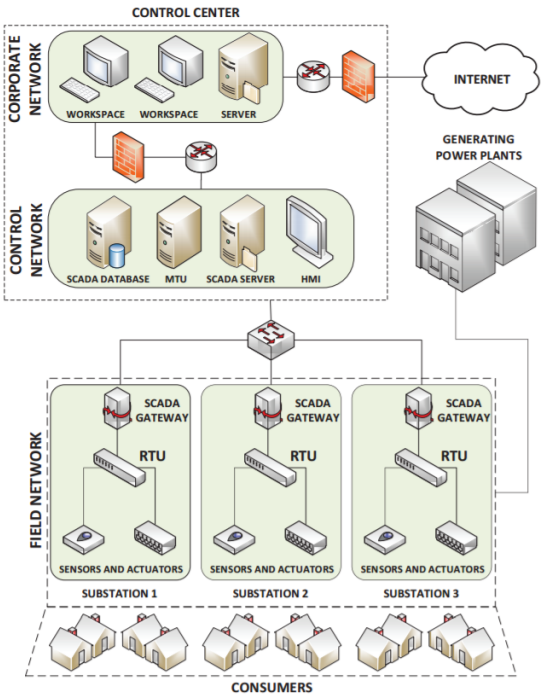


Figure 7.1: A typical SCADA system hardware architecture for smart grids [161]

The control center consists of a control and a corporate network. The corporate net-

work provides the operator with a Human-Machine Interface (HMI) that allows the access to system data, SCADA servers and databases that store operational and financial information. The SCADA server controls and gathers field information from geographically distributed substations or Remote Terminal Units (RTUs). As SCADA systems cover large areas, they use Wide Area Networks (WAN). Each substation manages power distribution in a given area. They consist of a Programmable Logical Controllers (PLCs) which are digital computers connected to a set of sensors and control actuators. PLCs also convert sensor data into digital data.

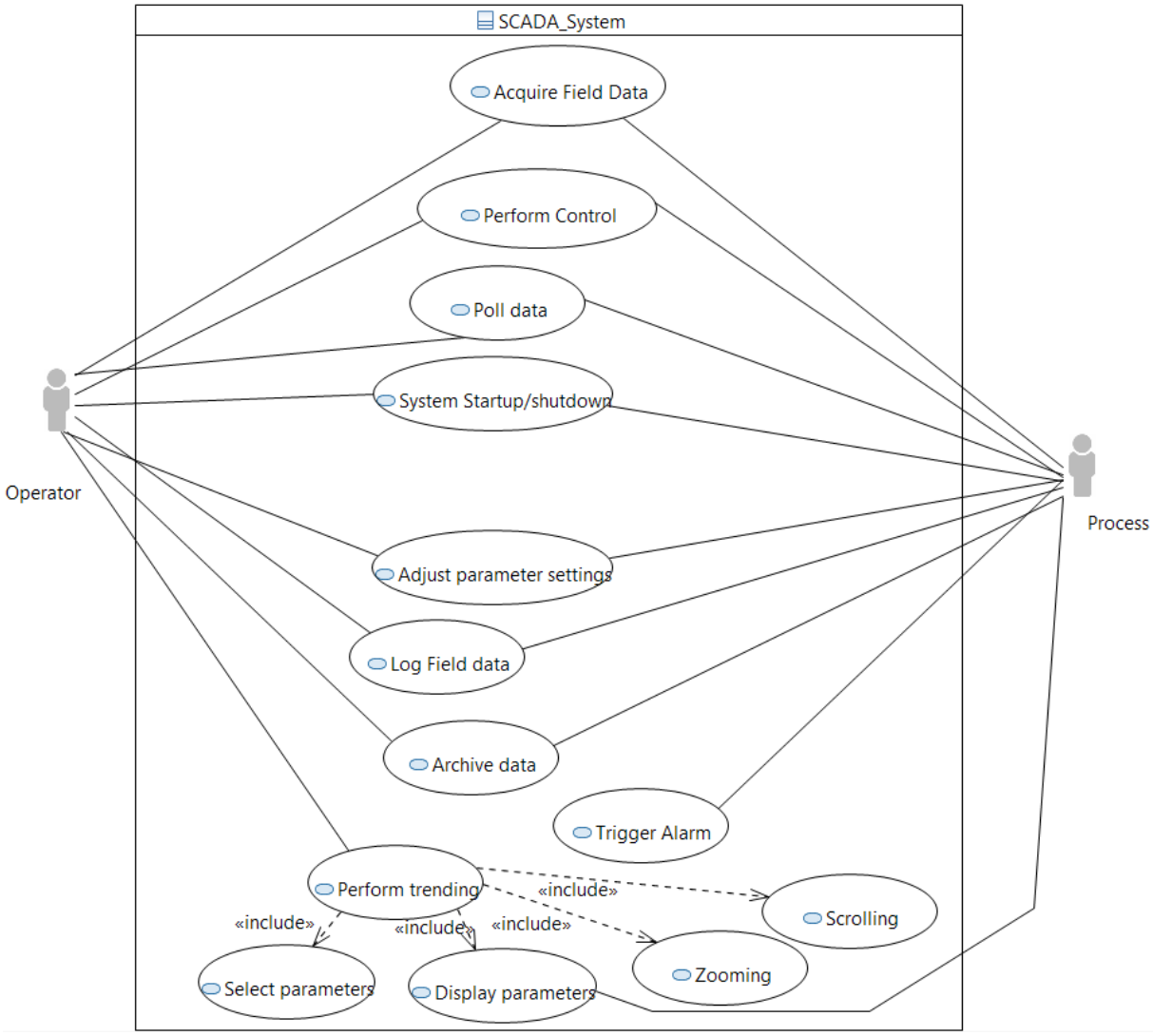


Figure 7.2: SCADA system use cases [35]

Figure 7.2 shows the use case diagram of the SCADA system. The use cases are the following [161]:

- Acquire Field Data
- Perform Control
- Poll Data
- System Start-up/shutdown
- Adjust Parameter Settings
- Log Field Data
- Archive Data
- Trigger Alarm
- Perform Trending
  - Select parameters
  - Display parameters
  - Zooming
  - Scrolling

SCADA security issues

Security is important for SCADA systems because of the high availability and reliability requirement of the critical infrastructure. It is sometimes very hard to shut down part of the system to change things or to repair them because they must be continuously available. Failure in SCADA systems can threaten life directly, or can have serious economic impact due the controlled critical infrastructure. It is very important for SCADA system to be safe and reliable. They have a good reputation in this field. However, the key issue nowadays is SCADA security. First generation of SCADA were introduced in the 1970's and second generation in 1980's. Many of these are still in operation especially second generations. They relied on two approaches for security:

- Security by isolation: based on the principle that if the system is not connected to the Ethernet then it cannot be attacked by external attackers. There is an Air gap which cannot be crossed. However it is still vulnerable to insider attacks.

- Security obscurity: based on the fact that SCADA systems used unusual programming languages and communication protocols. However this is also vulnerable to insider attackers who know about these technologies. In addition the documentation can be found on Internet or can be stolen.

Third generation SCADA systems use standard IT technologies and protocols (e.g., organizational wireless networking, Microsoft windows, TCP/IP and web browsers as interfaces). The third generation systems which are web connected are integrated with an interface to second generation systems. Internet-based SCADA is becoming the standard. The problems of connecting modern SCADA systems to the Internet that is not different from second generation systems is that their development was not thought in a security-aware environment. Air gap cannot exist anymore with modern SCADA system. In practice SCADA systems are connected to other systems in an organization and so information is passed. They are connected to maintenance systems provided by manufacturers. Workstations used by the operators as standard terminal are multipurpose machine that are Internet connected. In addition, operators still transfer information with USB drives. Hence, SCADA systems may be infected by malware from this. Because SCADA by large were not developed in security-aware environment, people are finding lots of security vulnerabilities (e.g., weak passwords, no firewalls, no input validation, unencrypted traffic through the Internet, etc.).

Security and Performance

The first challenge comes from the fact that SCADA systems are developed by domain engineers (e.g., power engineering). They generally have no background or training in software security techniques. Another challenge comes from the fact that it is often not possible to use standard security techniques. For instance, running checking systems in conjunction with SCADA system affects its performance in such way that process control is compromised. It may not be possible to install malware detection on some SCADA systems because of the lack of processing power of the system or because of the age of the used operating system. It is also difficult to take systems off-line because of the 24/7 availability requirements which would allow software security loopholes to be patched and maintained.

7.2.2 An Overview of the Model Repository Content

Before modeling and analyzing the case study, we give some information about the used model repository. The model repository contains so far :

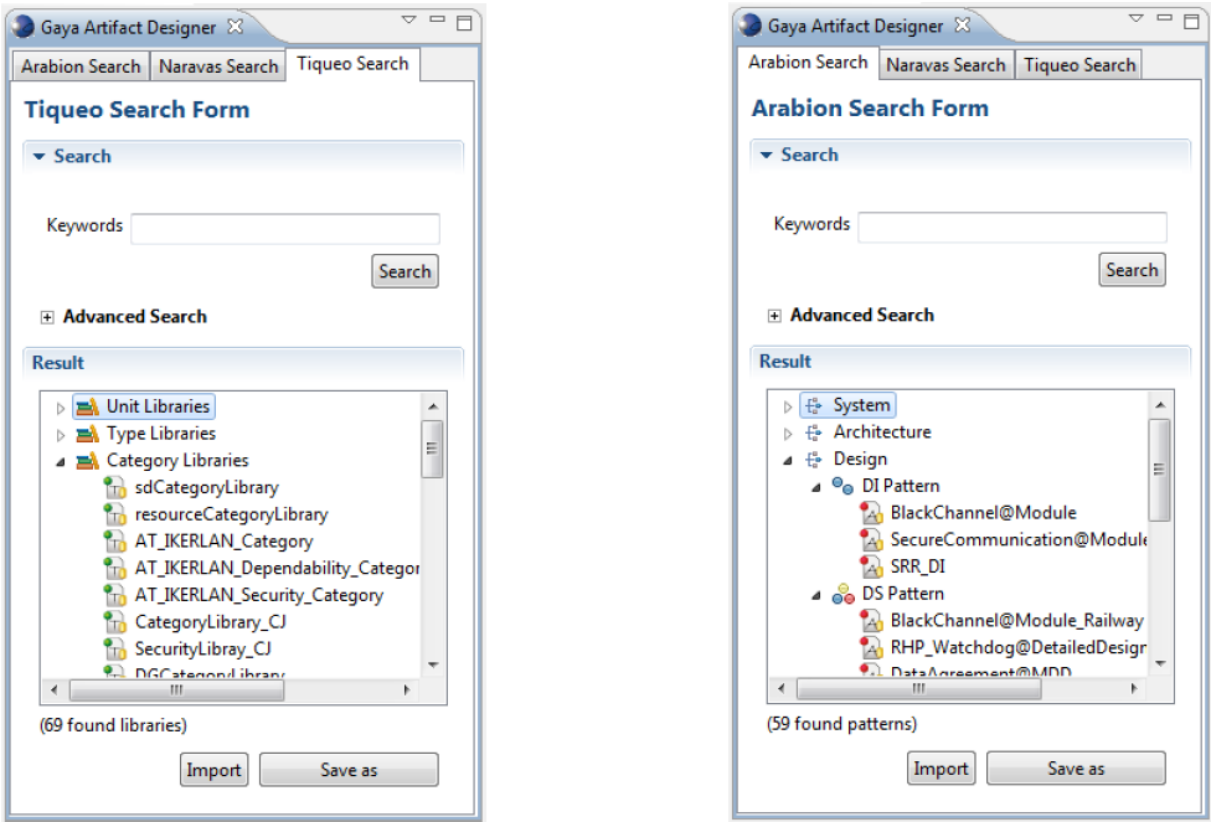


Figure 7.3: Model repository content: pattern and property models

- Property Libraries. 69 property model libraries (see the left part of Figure. 7.3):
  - 16 Unit Libraries
  - 23 Type Libraries
  - 20 Property Category Libraries
- Pattern Libraries. 59 pattern models (see the left part of Figure 7.3):
  - 20 System Level patterns (12 DI, 8 DS)
  - 25 Architecture Level patterns (9 DI, 16 DS)

7.2.3 Modeling the SCADA architecture

This section deals with the modeling of the SCADA system platform and software architecture using our tool chain. The models, are used to produce software architecture candidates which are evaluated with regards to formalized threat scenarios and real-time constraints.

System Architecture

The SCADA system architecture is composed of two system components as shown in Figure 7.4: Control center and RTUs that communicate. In addition, we show all the types used to model the component types.

Software Architecture and Platform

The software architecture and platform refines the system architecture. Figure 7.5 shows the software architecture model based on [35]. In addition, ports, interfaces, data types and transmitted messages are added in Figure 7.6 to provide a more detailed model of the application. The platform is modeled in Figure 7.7. We show the relationship between the system components and hardware nodes.

7.2.4 EBIOS Risk Assessment

Table 7.1 shows a list of feared events and threats causing them [48]. Each couple of (feared event, threats) constitutes a risk. Hence, three risks are identified.

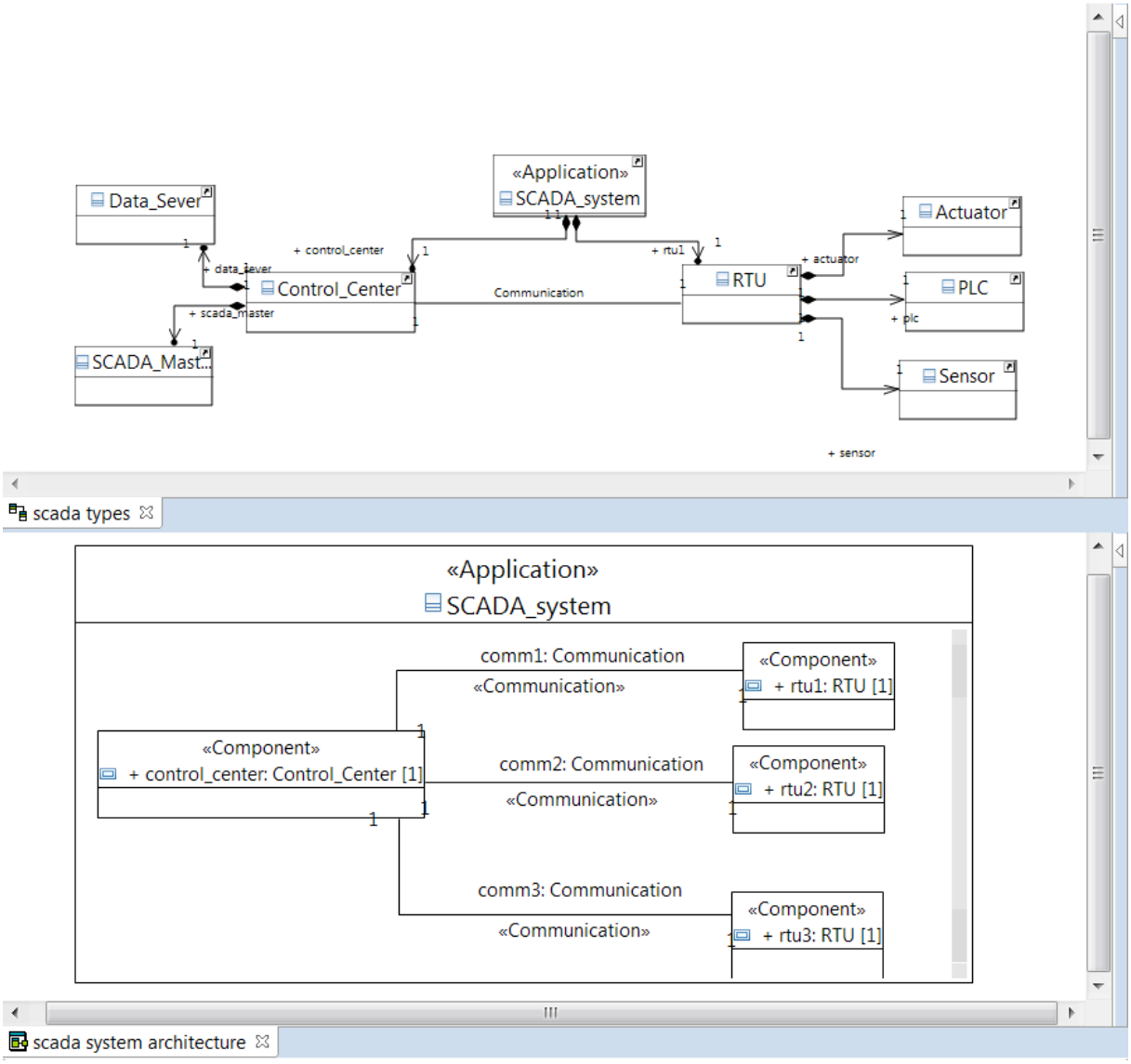


Figure 7.4: SCADA system architecture model

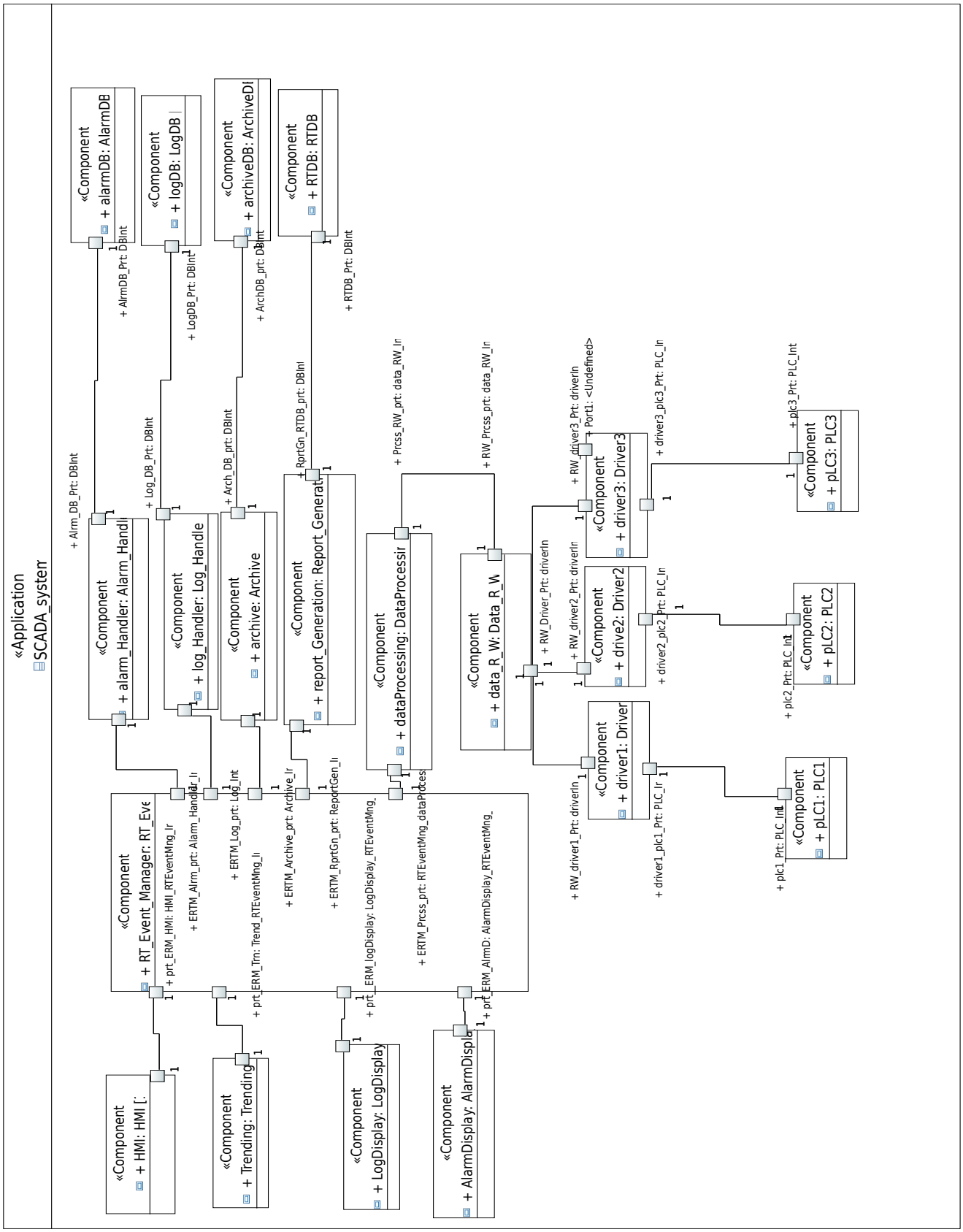


Figure 7.5: SCADA software architecture model

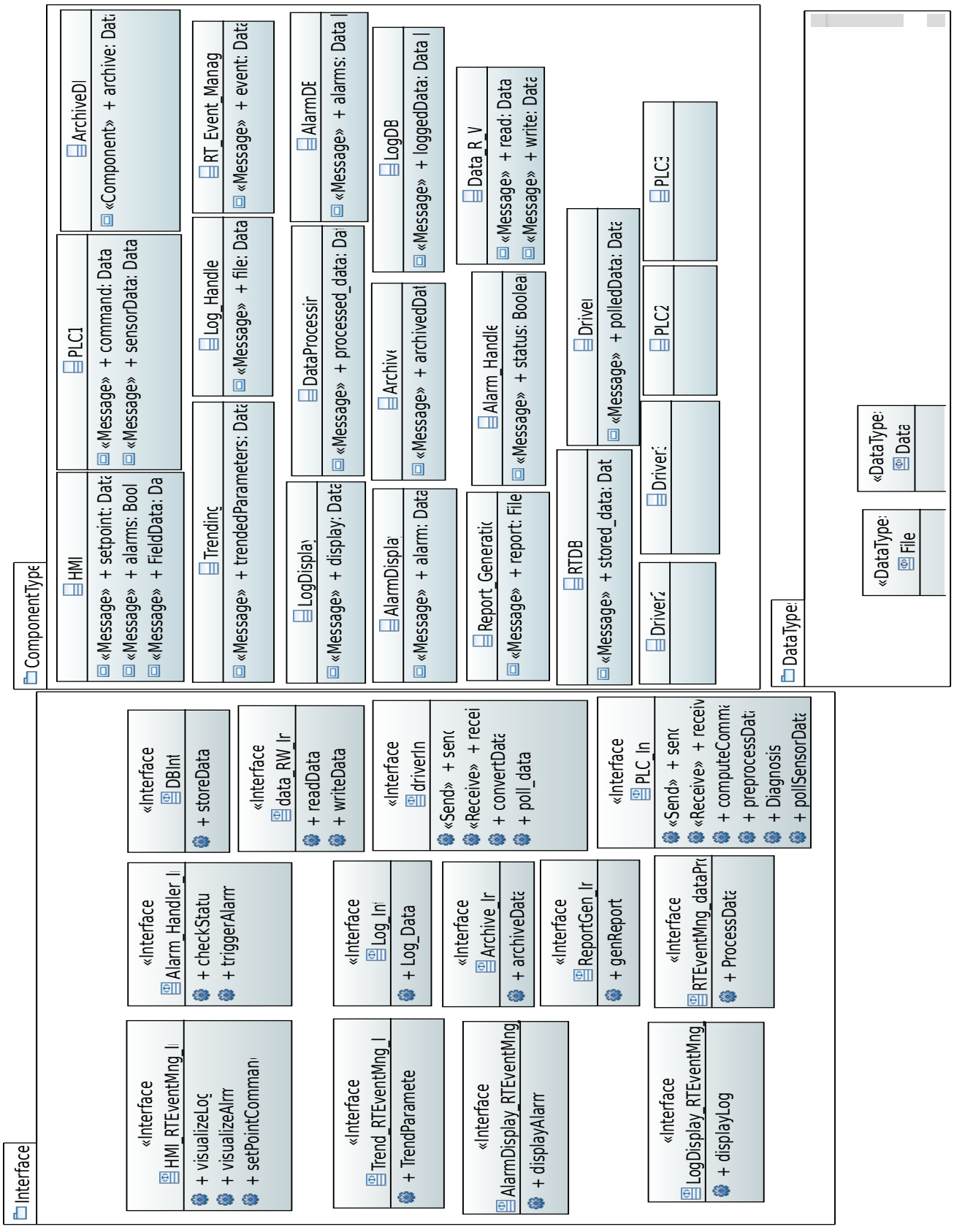


Figure 7.6: SCADA types and interfaces

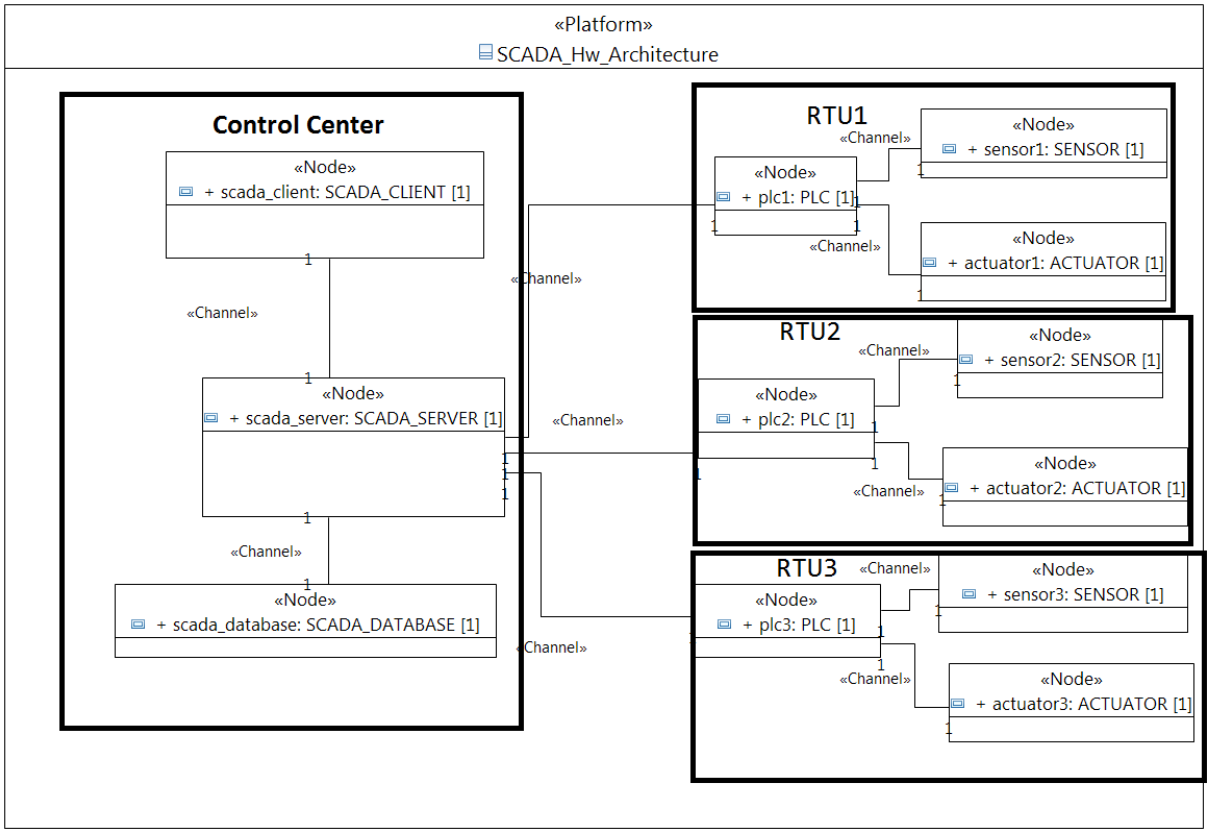


Figure 7.7: SCADA platform

Feared event	**Threats**	**Threat category**	**Property Category**
Control Center Disruption	**th1.** Physical attacks	Tampering	Integrity
	th2. Malicious settings of the field units	Tampering	Integrity
	th3. Wrong commands sent to the field units	Repudiation	Non-Repudiation
	th4. Malicious alteration of the parameters of the SCADA master	Tampering	Integrity
	th5. Denial of service attack	Denial of Service	Availability
Communication Disruption	**th6.** Sniffing commands	Information Disclosure	Confidentiality
	th7. Spoofing	Spoofing	Authenticity
	th8. Denial of service attack	Denial of Service	Availability
RTU Disruption	**th9.** Physical attacks	Tampering	Integrity
	th10. Malicious alteration of the run-time parameters	Tampering	Integrity
	th11. Incorrect commands sent to the central controller	Repudiation	Non-Repudiation
	th12. Malicious alarms sent to the central controller	Spoofing	Authenticity
	th13. Denial of service attack	Denial of Service	Availability

Table 7.1: SCADA system feared events and threats [48]

7.2.5 Selection and Instantiation

A System of Patterns is selected from the model repository protecting the system against the identified threats in Table 7.1. The System of Patterns is then instantiated in the modeling environment Papyrus (see Figure 7.8) thanks to *Semco4Papyrus* access tool.

The System of Patterns consists of a set of abstract patterns refined by concrete ones. The description of these patterns can be found in Appendix. A:

- Secure Communication: ensures that data passing across a network is secure. It can be refined by two alternative patterns:
  - SSL

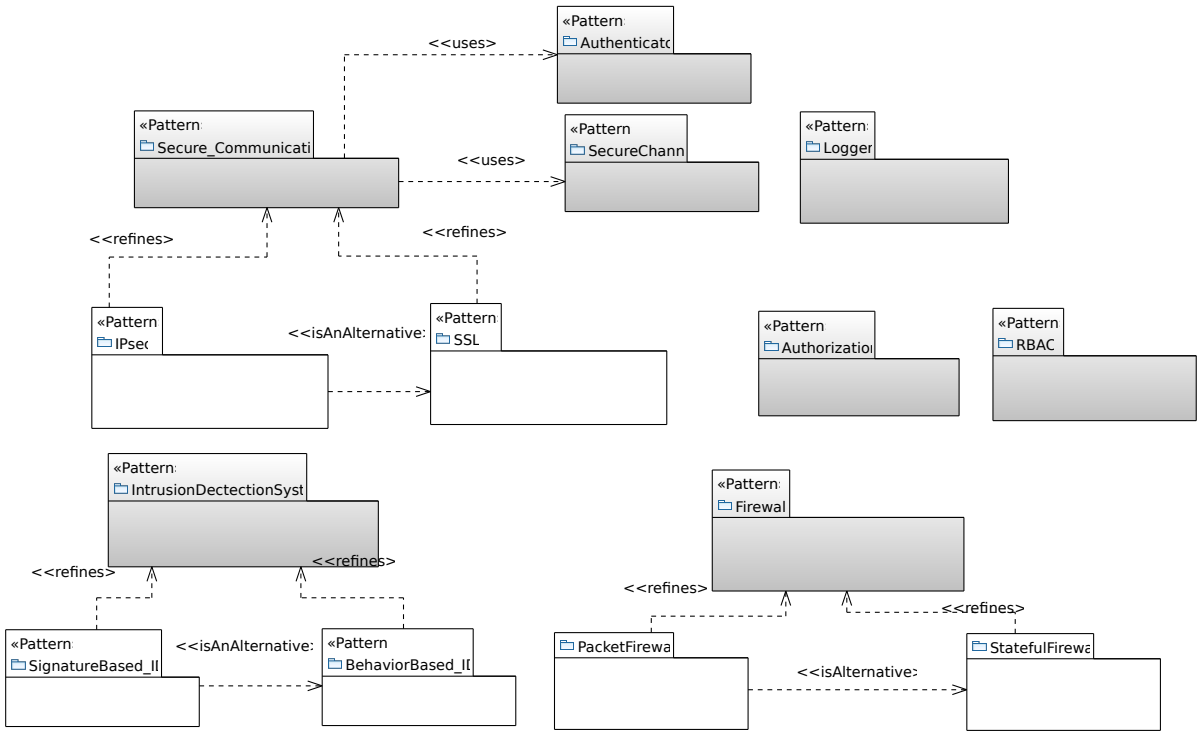


Figure 7.8: System of Patterns instantiated in Papyrus

  - IPsec

- Firewall: restricts access to internal network. It can be refined by the following alternative patterns:
  - Packet Firewall
  - Stateful Firewall
- Intrusion Detection System (IDS)
- Authorization
- RBAC
- Logger and Auditor

Using Table 7.1, the abstract patterns are linked with the SCADA system architecture to identify where they are used. This is possible because each feared event is relevant to a system component and each threat is stopped or mitigated with an abstract pattern. Figure 7.9 shows where the abstract patterns are used with regards to the system architecture.

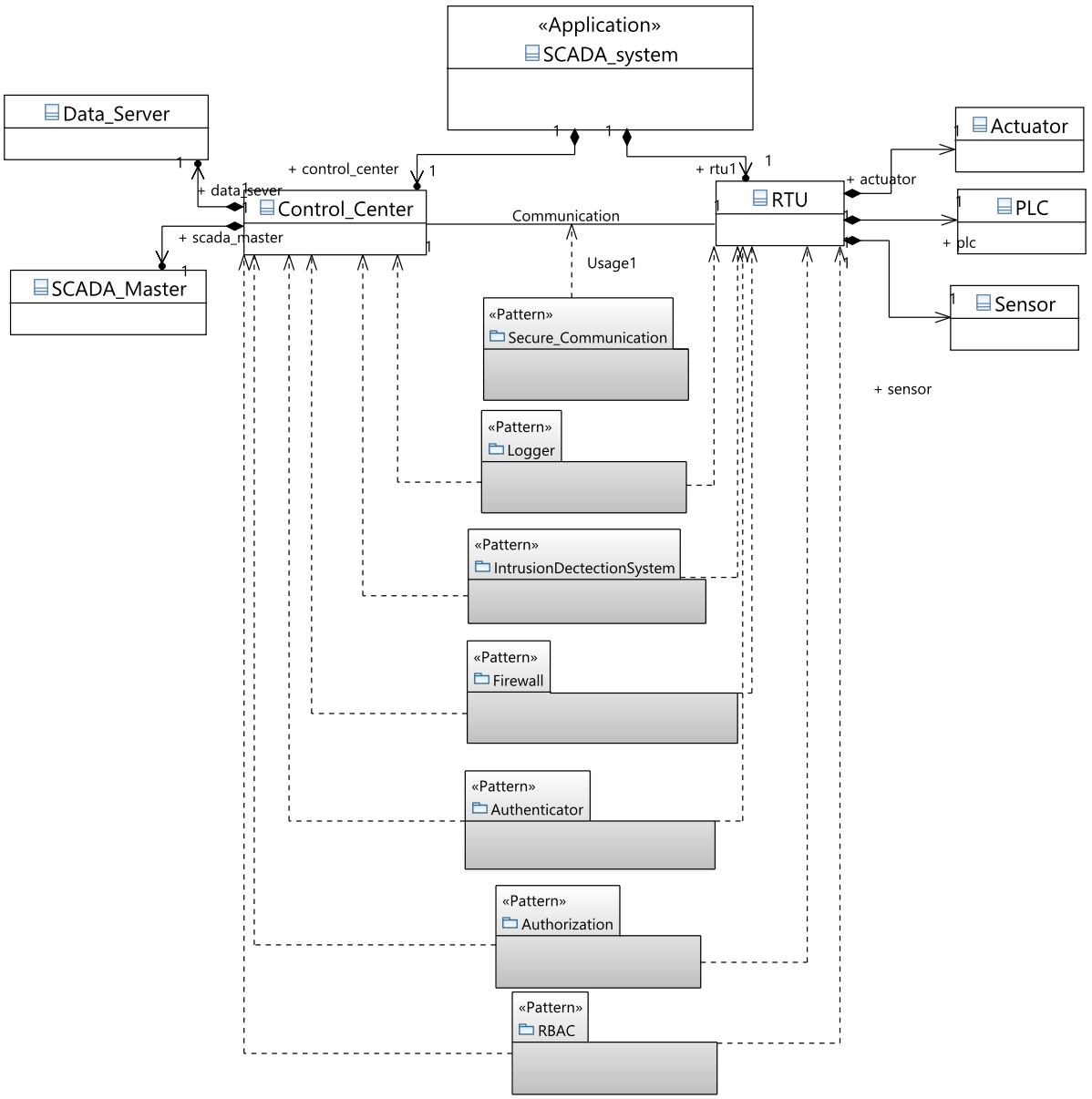


Figure 7.9: SCADA system architecture with pattern usage

Configurations	Concrete Security Patterns
Configuration 1	SSL, Packet Filtering
Configuration 2	SSL, Stateful Filtering
Configuration 3	IPsec, Packet Filtering
Configuration 4	IPsec, Stateful Filtering

Table 7.2: System of Patterns configurations

A System of Patterns configuration is a combination of concrete patterns (if any) refining an abstract pattern. Among the different System of Patterns configurations, we have selected four, described in Table 7.2, to be analyzed.

7.2.6 Pattern Integration

For each configuration, patterns are integrated into the SCADA system software architecture described in Figure 7.5. Thus, we obtain software architecture candidates 1, 2, 3 and 4 for each pattern configuration 1, 2, 3 and 4. Figure 7.10 shows software architecture candidate 1. The integration aims at protecting: (1) the communication between the SCADA server and PLCs against information disclosure and spoofing (e.g., Man-In-The middle attacks), (2) the SCADA server and PLCs against Denial of Service attacks. The integration adds security mechanisms relevant to the SSL pattern (protocol controller, encryptor, decryptor, signer, verifier, key exchange, authenticator) and one Packet Filtering pattern security mechanism of the same name.

7.2.7 Software Threat Analysis

The software architecture candidates and the initial software architecture are submitted to threat analysis. Table 7.3 and Table 7.4 shows the number of threats per threat category before and after the integration of security patterns. The number of threats in the four software architecture candidates is the same because, the concrete patterns alternatives provide the same security property. There are no Man-In-The-Middle and Tampering (during transmission) threats after the integration because of the application of SSL pattern that guarantees confidentiality and integrity of messages between the SCADA server and the PLCs. Four Denial of Service and Injection threats remain because four software components (HMI, Trending, LogDisplay and AlarmDisplay) have public ports and there is a lack of Firewall and Authorization mechanisms.

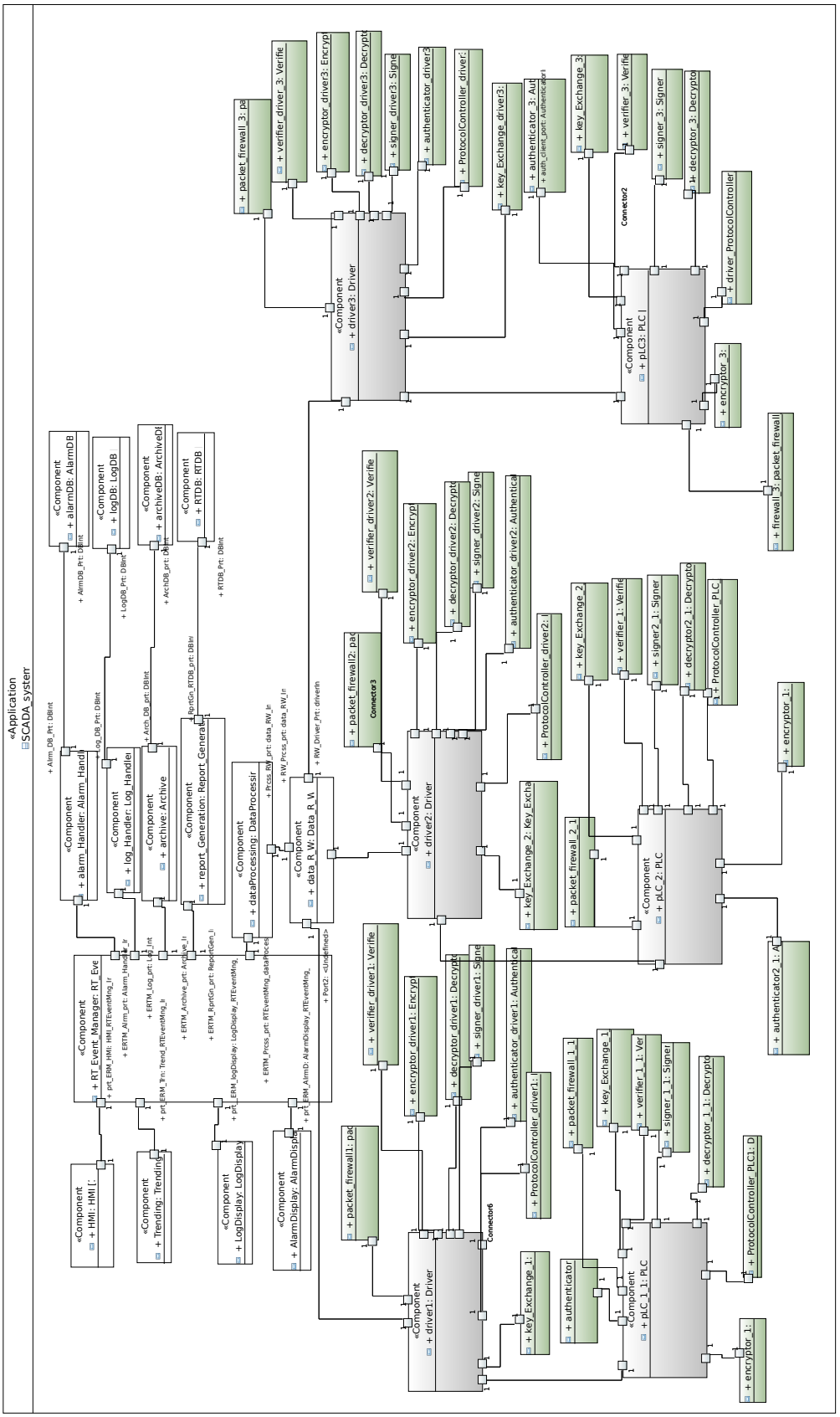


Figure 7.10: SCADA software architecture after the integration of configuration 1

Threat categories	Detected threats
Man-In-The-Middle	6
Tampering	6
Denial of Service	10
Injection	10
Total	24

Table 7.3: Detected threats per threat scenario before pattern integration

Threat categories	Detected threats
Man-In-The-Middle	0
Tampering	0
Denial of Service	4
Injection	4
Total	8

Table 7.4: Detected threats per threat scenario after pattern integration for the four security pattern-based software architectures

7.2.8 Real-time Analysis

In this step, The aim is to keep only the architecture candidates that satisfy real-time requirements. We evaluate software architecture candidates 1, 2, 3 and 4 with regards to real-time constraints. In the analysis, we consider the transactions between the *SCADA Server* and the *PLCs*. Since the three PLCs are similar the study focuses on one PLC. For the case study, we have used the following techniques:

- Scheduling policy : *Fixed offset-based scheduling with Rate-Monotonic Analysis (RMA)* [153].
- Task partitioning and allocation : Deterministic (static) Task Partitioning strategy.

Figure 7.11 shows the end-to-end flows before pattern integration and the deployment on the platform. These end-to-end flows are modeled according to messages in the architecture. The partitioning of end-to-end flows into tasks and allocation of tasks into nodes are also shown. Figure 7.12 shows the generated task model that consists of seven tasks with timing parameters. The values of the SCADA system timing parameters are based on IEEE 1646 standard [3] specifying communication deadlines and IEC 61850 [77] specifying communication network delays in different information categories. Functions WCET, the assigned tasks and nodes are shown in Table 7.5.

Security pattern function WCETs and their assigned tasks are shown in Table 7.6. Alternative concrete patterns have the same functions but have different WCETs. The

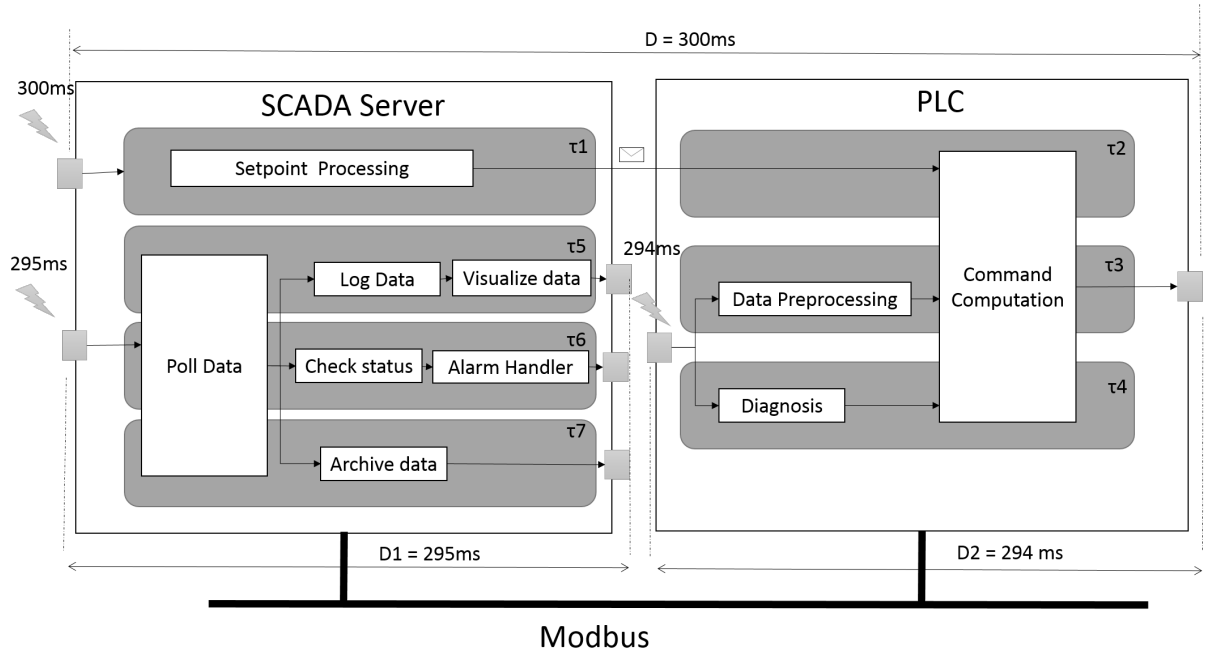


Figure 7.11: End-to-end flows and deployment

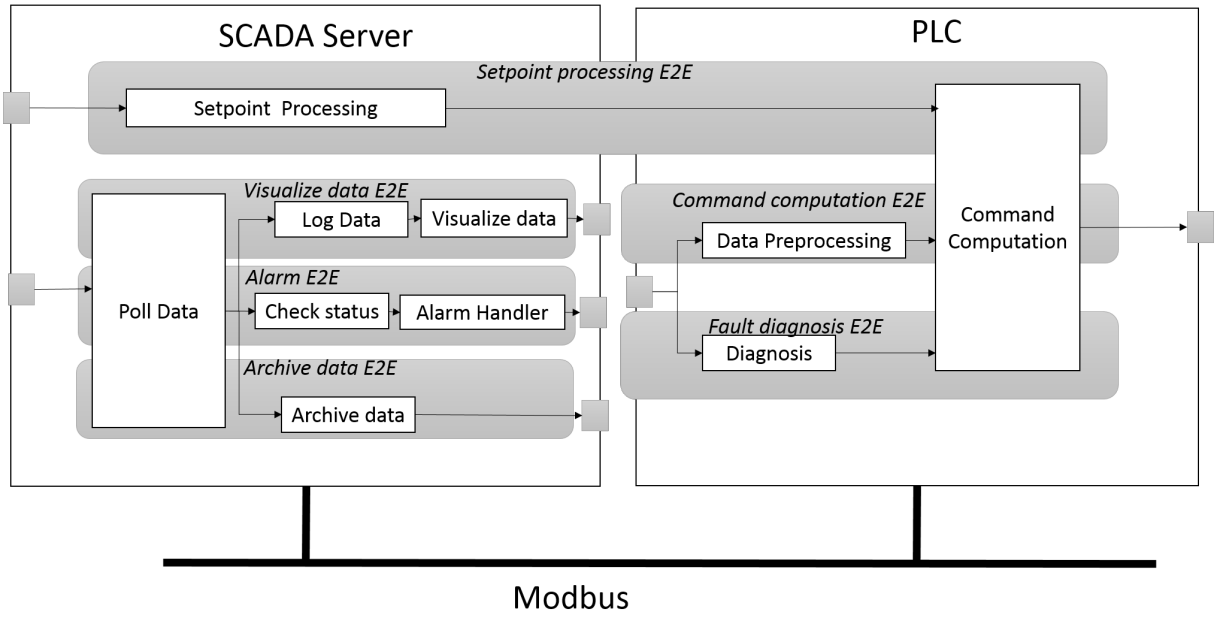


Figure 7.12: Task partitioning and allocation

Functions	WCET	Task	Node
Setpoint Processing	8.7	$\tau 1$	SCADA Server
Poll Data	9.6	$\tau 1$, $\tau 6$, $\tau 7$	SCADA Server
Log Data	8.5	$\tau 5$	SCADA Server
Check Status	9.6	$\tau 6$	SCADA Server
Visualize Data	10.5	$\tau 5$	SCADA Server
Alarm Handler	10.3	$\tau 6$	SCADA Server
Archive Data	9.5	$\tau 7$	SCADA Server
Command Computation	10	$\tau 2$, $\tau 3$, $\tau 4$	PLC
Data Preprocessing	9.5	$\tau 3$	PLC
Diagnosis	8.9	$\tau 4$	PLC

Table 7.5: Timing parameters and deployment of SCADA functions

timing parameters are based on a review of technical reports of SSL/IPsec [11], and stateful/packet firewall [71].

Patterns	Functions	WCET		Task
		(1)	(2)	
SSL (1) IPsec (2)	Authentication	9.7	38.7	$\tau 1$
	Key exchange	10.1	39.6	$\tau 1$
	Encryption	9.9	9.9	$\tau 1$
	HMAC	9.2	9.2	$\tau 1$
	Decryption	10.3	10.3	$\tau 2$
	Integrity checking	10.2	10.2	$\tau 2$
Packet Filtering (1) Stateful Filtering (2)	Filtering	10	40	$\tau 7$

Table 7.6: Timing parameters and deployment of security pattern functions

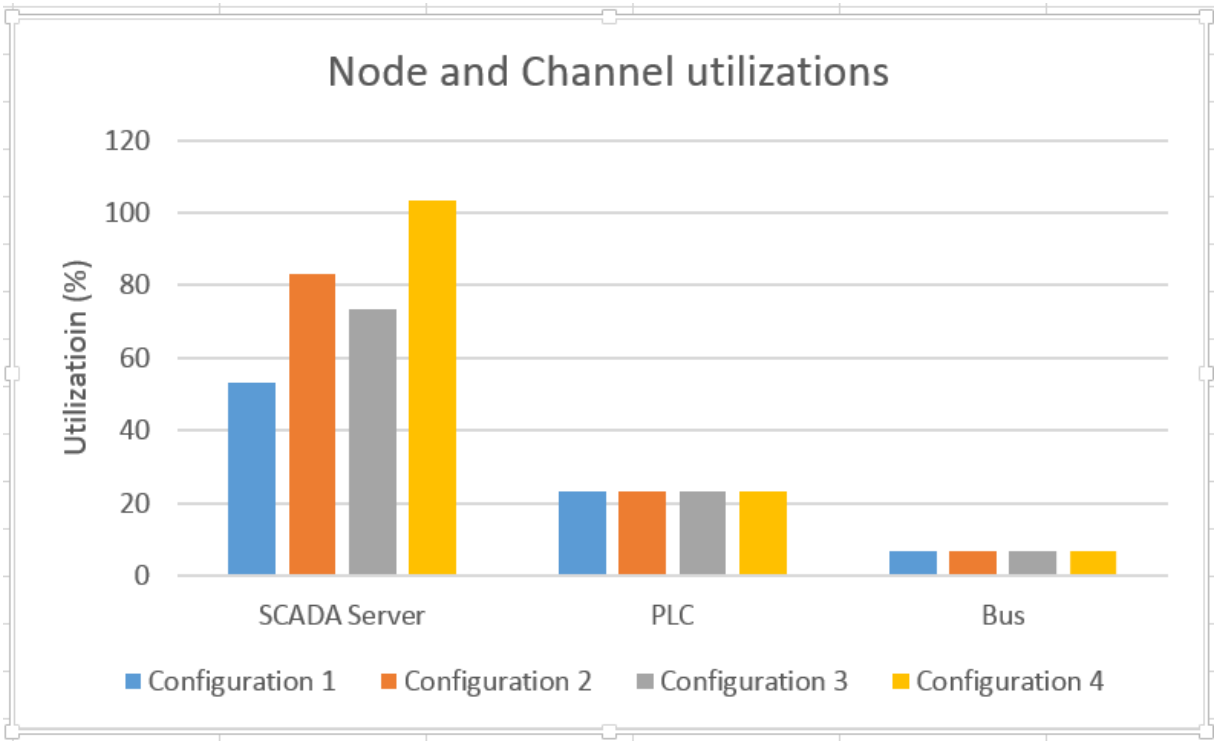


Figure 7.13: Node and channel utilizations

Figure 7.13 shows node and channel utilization results of the four software architecture candidates. Candidate 4 is rejected because the SCADA master utilization 103.33% exceeds the threshold (100%). This is because authentication and key exchange functions in SSL have higher processor demands then in IPsec. In addition stateful filtering is more demanding then packet filtering. Response time analysis is performed for candidates 1, 2 and 3 since they pass the processor utilization test. Figure 7.14 shows task WCRTs for each software architecture candidate 1, 2 and 3.

7.3 Feasibility of the approach

We study the feasibility of the approach by analyzing the results obtained on the SCADA system case study. We show the feasibility of the approach through the execution of three activities: (1) threat analysis - we compare the detected threats to threats obtained with an attack scenario simulation framework named ASTORIA [161]; (2) pattern integration - we compare threats before and after security pattern integration; (3) Real-time evaluation - we discuss the benefits of such evaluation at early stage.

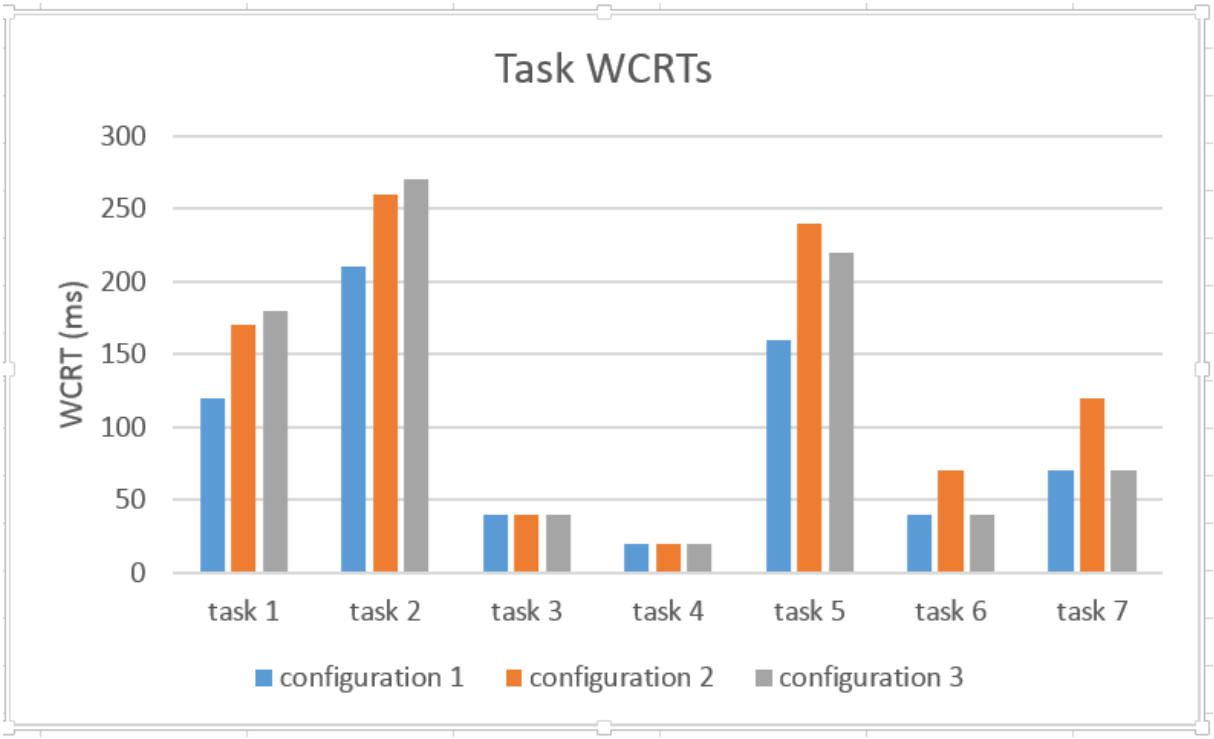


Figure 7.14: Task WCRTs

7.3.1 Software Threat Analysis

To assess the threat analysis module, we compare the results obtained to the work in [161]. The paper proposes a framework named ASTORIA [161] for attack scenario simulation for smart grid systems. The selection of the framework was motivated by the fact that ASTORIA is a simulation framework whereas ours is a formal verification-like framework. The ASTORIA team have simulated attack scenarios and evaluated their impact on the smart grid system to discover existing threats. Table 7.7 presents the results obtained with ASTORIA framework and our threat analysis framework.

For each asset, we conclude that all the detected threats are TPs i.e., all detected threats are threats. In addition, our framework detected at the level of RTUs and communication new threats i.e, Denial of Service and Tampering. FNs, i.e., threats that were not detected are due to different reasons:

- Some attack scenarios were simply not formalized or out of scope of our framework. For instance, Phishing is an attack scenario that attempts to obtain sensitive information such as credentials, and credit card details for malicious reasons using an electronic communication generally emails. This attack exploits social engineering was which is out of scope the study.

Assets	**Threats with ASTORIA [161]**	**Detected threats with our framework**
Control Center	Malicious Software Phishing Ping sweeps Port Scanning Spyware **Injection** Unauthorized access Spoofing Replay **Denial of Service**	**Injection** **Denial of Service**
RTU	Malicious Software Phishing Ping sweeps Port Scanning Spyware Unauthorized access Spoofing Replay **Denial of Service**	**Injection** **Denial of Service**
Communication	Eavesdropping Sniffing **Man-In-The-Middle** Denial of Service Replay	**Man-In-The-Middle** **Tampering**

Table 7.7: Threat Analysis results comparison

- Some attack scenarios are of the same kind or are consequences of some formalized threat scenarios. For instance, replay attacks are a kind of Man-In-The-Middle attacks where the attacker maliciously or fraudulently repeats or delays a valid data transmission. Ping sweeps are generally used to check if a node is alive or dead. It can be used as flooding technique and becomes a Denial of Service attack.
- Some attack scenarios are at a lower stage (implementation) such as malicious software. In fact, we deal with software architecture analysis and not with code analysis.

7.3.2 Pattern Integration

The pattern integration process is evaluated via software threat analysis. Indeed, Table 7.3 and Table 7.3 show threats before and after pattern integration respectively. We conclude

that the patterns that were selected have accomplished their task by decreasing the threats related to the threat categories that they were selected for. However, since we have not formalized other attack scenarios, it is very likely some threats are left unmitigated. This issue should be investigated as a future work. Another form of assessment is due the fact that the pattern brings a certain security property. We have shown that the result of the integration verifies this property in chapter 4.

7.3.3 Real-time Analysis

Incorporating security during architecture design may produce different architecture candidates. If all these alternatives are expected to be correct regarding security issues their impact in terms of other non-functional requirements such as real-time constraints, memory or energy consumption should also be considered to select a solution offering a good trade-off. In this work, we have focused on real-time analysis of architecture candidates illustrated through a SCADA system with strong real-time and security needs. This work can be beneficial to resource constrained embedded systems e.g., automotive, avionics. For instance in EAST-ADL [37], trade-off analysis is performed for one design model with different parameters whose values determine whether the design satisfies multi-concern objectives. Our work adds a step forward which is the analysis of different design alternative models against non-functional concerns (security in our case). One important point is that the experiment has required some effort in quantifying real-time parameters of security pattern security functions. Some function WCETs were estimations. For example, HMAC function may have different WCETs as it depends on the used algorithm (e.g., HMAC-SHA-1-96, HMAC-MD5). However, we believe that estimations are enough as the approach is meant for high level evaluation and architecture decision making. For example, if none of the architecture candidates respected the real-time requirements because of overload; the architecture of SCADA can be rethought leading to adding an execution node. However, if there is a big gap in timing parameters for the same security functions, estimations can affect the soundness of the results. The solution in this case is to define timing configurations which are possible by defining upper and lower bounds using MARTE. Hence optimization heuristics (supported by "Qompass Architect") can provide an optimal architecture candidate. This point needs further investigations and is left as a future work.

7.4 Conclusion

In this chapter we have assessed the feasibility of the contributions (from **RG1** to **RG4**) through the modeling and analysis of a SCADA system case study with strong demands in security and responded to **(RG5)**.

SCADA systems are interesting case study because security is a high-critical issue and they are different from classical ICT system such as the web application working example used for the illustration. We have used Domain Specific Languages to model the software and hardware architecture of the system. Once the modeling has been performed, risk assessment was applied to identify potential system threats. These threats were categorized in order to select a number of security patterns constituting a System of Patterns. The resulting security architecture candidates were analyzed according to real-time constraints and formalized threat scenarios.

The results obtained from each contribution were analyzed to assess the feasibility of the approach. We have assessed the threat analysis framework, discussed in chapter 5, by comparing it to the attack scenario simulation framework. The integration process, discussed in chapter 6, was assessed with regards to the detected threats before and after pattern integration. Finally real-time analysis in chapter 4 was assessed by discussing the benefits that it can give to a number of domains such as avionics and automotive, the potential improvements by using existing work and the difficulties and efforts to perform such analysis. According to the results of the assessment, the overall feasibility of the different contributions of the approach is given. Now, this approach needs to be assessed according to the acceptance of software architects. As a future work, we intend to assess the acceptance of the approach by performing a survey. This point is detailed in future works.

Chapter 8

Conclusion and Future Work

Contents

8.1 Summary and Contributions

In this thesis we propose a framework for engineering secure software architectures through modeling techniques, a set of devoted analyzes components and the use of security patterns. The proposed framework consists of several features to assist architects in building secure architectures using a repository of secure solutions models. The contributions of this work are threefold: (1) an integrated design framework for the specification and analysis of secure software architectures, (2) a novel model- and pattern-based methodology and (3) a set of supporting tools.

The approach consists of the following steps (a) model-based risk assessment performed on the architecture to identify threats, (b) selection and instantiation of security pattern models towards the modeling environment for stopping or mitigating the identified threats, (c) correct integration of security pattern models into the architecture model, (d) model-based analysis of architecture candidates with regards to real-time requirements and residual threats. The aim is to compare these architecture candidates and keep only the ones satisfying real-time constraints. The detection of residual threats is done by validating the architecture model against formalized threat scenarios from STRIDE and based on existing threat references (e.g., CAPEC).

8.1.1 Integrated Design Framework

We have defined an integrated design framework, using UML profiles, for the specification and analysis of secure software architectures (to answer **RG1**):

- **EBIOS UML profile.** EBIOS concepts [61] are used to create a UML profile in order o represent risk assessment artifacts.
- **Architecture specification.** We use the extension mechanisms of UML [130] and create a UML profile for component-based software architectures. The profile is then augmented with verifiable constraints, written in the Object Constraint Language (OCL), that help system architect in systematically identifying security threats.
- **Pattern specification.** The SEPM metamodel [61] is used to create a UML profile in order o represent security patterns. This UML profile has been developed to allow the instantiation of patterns in the targeted modeling environment.
- **Integration.** In order support pattern integration methodology, we create a UML profile that helps architects in relating the concepts of the solution provided by the pattern to those of the target architecture.

8.1.2 Model- and Pattern-Based Methodology

In chapter 4, we extend previous work of the team [66] and [135] which provide a first solution of design pattern integration in the context of object-oriented applications and safety requirements respectively. We present a process for the selection, instantiation and integration of proper security patterns during architecture design based on risk assessment recommendations (to answer **RG2**). The modeling is based on accepted OMG standards: UML, its extension mechanism (Profiles) and OCL to express constraints. Our main goal is to develop a way to select relevant security pattern models from a patterns repository (SEMCO in our case) according to recommendations issued by a risk analysis of an architecture model. These patterns are instantiated on the fly under the form of compatible UML models and then integrated into the application to build a secured software architecture candidates.

In chapter 5, we discuss threat analysis of the software architecture candidates after the application of security patterns with regards to residual threats (to answer a part of **RG3**). Indeed, the selected System of Patterns stops or mitigates specific system threats derived from risk assessment. However some system threats have been accepted and other software threats may appear due to the level of detail of the software architecture model.

The goal is a holistic exploration of application software architectures for the elicitation of potential threats. In order to do this, we show the formalization process of a set of threat scenarios in OCL language.

In chapter 6, we extend previous works of the team [121, 62]. In [121], a MARTE-based framework for real-time schedulability analysis at early design stages was presented. In [62], an approach to support Security, Dependability and Resource Trade-offs using Pattern-based Development and Model-driven Engineering was presented. Here we go one step further, we propose a process and its tool support for the analysis of architecture candidates against real-time requirements (to answer a part of **RG3**).

8.1.3 Tool Support

Furthermore, we walk through a prototype supporting the approach (to answer **RG4**). In our context, we use the Eclipse Modeling Framework and its accompanying modeling tools developed around Papyrus. Currently the tool is composed of different modules provided as Eclipse plugins. An example of scenario of using the proposed integrated set of tools is visualized in Figure 8.1. Note, however, that our vision is not limited to the Eclipse platform. Here, we outline the different Eclipse tools used in the development of the proposed tool suite. Among the tools used here are cited:

- **Papyrus** is a modeling and analysis environment. This tool has been described in section 2.5.2. In our context, it is used to model the architecture which is then submitted to Sophia (1).
- **Sophia** is integrated in Papyrus and is dedicated to safety of systems described in UML or its extensions. We have contributed to the extension of Sophia for security analysis.
- **SEMCOMDT**[1] is an environment for the specification and development of a reuse model repository [63]. This tool has been described in section 2.5.3.
- **Qompass Architect** [92] is integrated in the Papyrus environment. It is a model-based tool developed in CEA LIST for QoS assessment and optimization of real-time architectures. Qompass Architect explores non-functional properties of real-time architectures to finally synthesize an optimized architecture.

To evaluate the proposed approach to support the modeling and analysis of secure architectures with patterns, we developed the following modules:

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

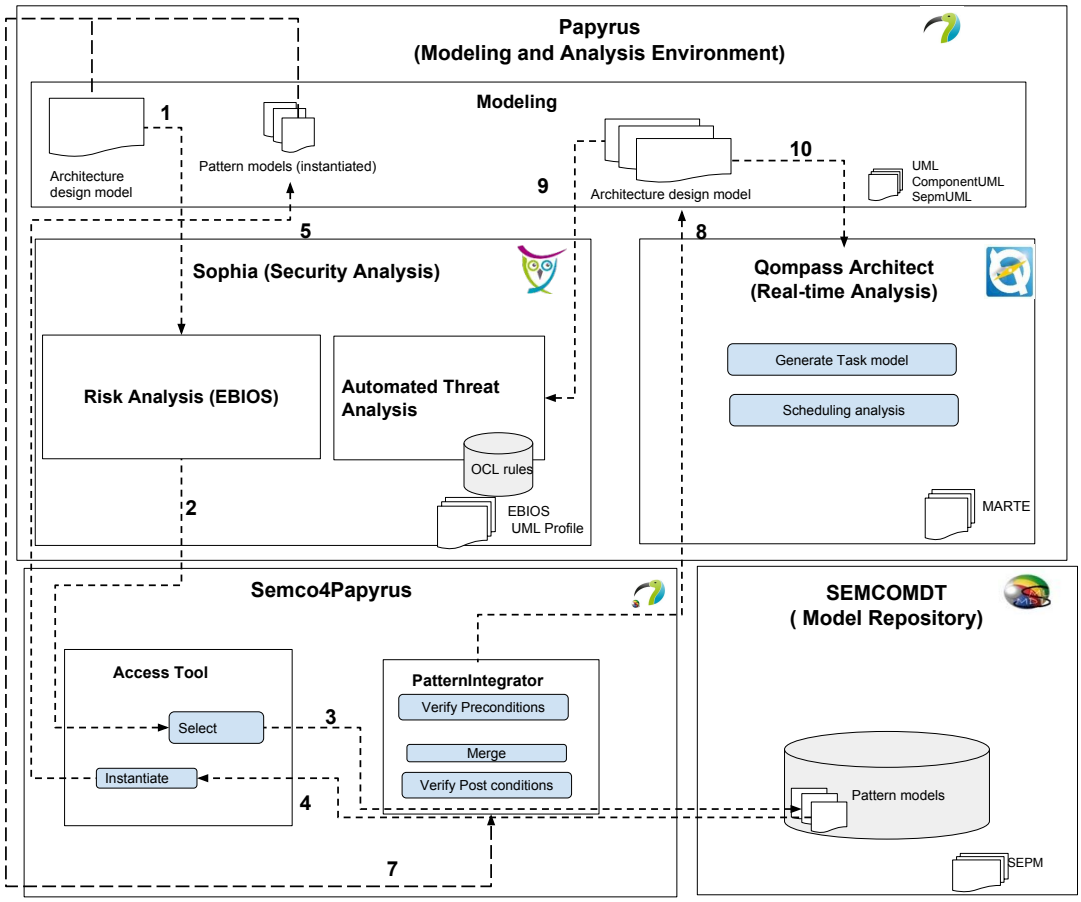


Figure 8.1: Tool-flow of the MDE-tool suite

- **EBIOS Risk Analysis** enables the modeling of feared events, threats and risks. It allows the automatic computation of risk level from likelihood and impact levels. These risks can be categorized according to security property categories and threat categories. Once risks are categorized, this information serves as input for Semco4Papyrus (2) to select patterns and instantiate them.

- **Automatic Threat Analysis** enables a holistic exploration of the software architecture and detects threats according to formalized threat scenarios. The current version supports four threat scenarios: Man-In-The-Middle, Tampering, Injection and Denial of Service. The results allow the revaluation of risks. If not risks are acceptable, the architecture is rejected.

- **Semco4Papyrus** is composed of two modules:

  - **Access tool** is a GUI exposed to the end user. It enables selection (3) and instantiation (4) of patterns stored in a SEMCOMDT based on EBIOS risk analysis results. The instantiation is performed with a Model-To-Model transformation towards Papyrus (5).

  - **PatternIntegrator** supports correct pattern integration in Papyrus and provides a generic scheme for pattern integration instead of ad-hoc transformation. It is based on merge and verification techniques. Instantiated patterns together with the architecture design model are submitted to PatternIntegrator (6). The results are architecture candidates (7). Each architecture candidate is submitted for Automated Threat Analysis (8). If the candidate is accepted, it is submitted to Qompass Architect (9).

8.1.4 Assessment

The proposed approach, its modeling framework and its tool support have been applied to a SCADA system case study in Chapter 7. Following the description of the use case and its implementation using the proposed framework, we have assessed the feasibility of the contributions of our work (from **RG1** to **RG4**) through the modeling and analysis of a SCADA (Supervisory Control And Data Acquisition) system **(RG5)**. We discuss the assessment of the results for each contribution. The selection of an application for the proposed approach has to be considered as a very important decision for our work. It has a direct impact on the selection of patterns, and thus on the whole assessment.

8.2 Limitations and Future Work

After having analyzed the case study and the conducted assessment, we identified lacks in the current version of our approach and defined future work objectives to enhance the proposed approach.

The limitations and future work consider different parts of the approach, namely (1) risk assessment, (2) pattern selection, (3) pattern integration, (4) model-based real-time analysis, (5) model-based software threat analysis and (6) the assessment. In the following section we detail the identified limitations and give ideas of possible solutions:

1. *Risk assessment.* Risk assessment can be improved by automating the identification of system threats. This is possible by reusing the work of software threat analysis. In fact, software threat analysis is used to detect threats at the software architecture design level according to formalized rules. The formalization of rules describes the context in which a certain threat may exploit an absence of one or more security mechanisms, their weakness or their incorrect integration. Therefore, at the level of system architecture, the formalization of system threats may be simplified to the non-existence of security mechanism.

2. *Pattern selection.* Pattern selection results can be improved by specifying the type of components where to apply the pattern. For example between a client and server. This can be done by reusing the work of [73].

3. *Pattern integration:*

   - *a. Role binding.* Similarly to 2, the work of [73] can be used to propose bindings. Indeed, once a pattern is proposed, bindings can be proposed to the user according to the type of components where we want to apply the pattern .

   - *b. Behavior.* During the integration we have considered a structural view of the architecture represented in UML composite diagram. We also need to consider behavior during the integration process. Currently, we consider messages as part of the structure of the solution. Hence, messages should be represented in a dedicated diagram such as the UML information diagram that takes into account the order of messages. A starting point can be the work of [40] where the authors studied the composition of security patterns behaviors modeled in a sequence diagram.

4. *Model-Based Real-Time Analysis.* We have considered estimations during real-time analysis. However, if there is a big gap in timing parameters for the same security functions, estimations can affect the soundness of the results. The solution in this case is to define timing configurations which are possible by defining upper and lower bounds using MARTE. The current version of the tool "Qompass Architect" can perform real-time analysis for a model with timing configurations as input. In this case, for one architecture candidate with timing configurations, node utilizations and response-time would have an upper and lower bound. Hence optimization heuristics (supported by "Qompass Architect") can provide an optimal architecture candidate. In addition, this work can benefit from [37] by extending trade-off analysis for different design alternative models instead of design space exploration for one design model. After the analysis of the secure architecture candidates is done and if multiple solutions remain, the "best" solution (if any) can be looked up by quantifying security and real-time objectives (e.g., number of threats of the architecture candidate as security objective and response time of an end-to-end flow as a real-time objective). Hence, these objective may be optimzed.

5. *Model-Based Software Threat Analysis:*

   - Threat analysis can be generalized by replacing ComponentUML with OMG standards for Component-Based Development particularly UCM [131]. The second step is to construct a library of helpers to easily formalize threat scenarios.
   - The specification of threats was done using OCL. OCL is a general language for constraining UML models. We want to enable security experts to contribute to the threat knowledge-base, who are not necessarily familiar with OCL, with less effort. In this context, we can inspect DSMLs for specifying these rules and then study mappings towards OCL. In literature we can find a number of Existing DSMLs for security specification (requirements, properties) [26, 105] which can be a starting point.

6. *Assessment.*

   In the near future, we plan to conduct an experiment in which we will present the approach and the solution of our case study to collect feedback from industry practitioners through a survey. In particular, we wanted to assess the perception of using patterns coupled with the modeling approaches to engineering secure systems. Duplicate the study to address secure software system development in other domains

and perform the survey with other subjects (e.g., students) have been considered also as future work.

8.3 Perspectives

Perspectives emerging from this thesis are manifold. These perspectives are long-term objectives and consist of enhancements related to (1) enabling traceability during pattern integration process, (2) performing interplay between security and safety, (3) linking the approach with the implementation phase and (4) finally to managing the implemented and deployed system.

1. *Traceability.* It can be useful to keep traceability links between patterns that have been integrated and where they have been applied in the architecture. This can be interesting when justifying that risks of architecture have been mitigated. For this end, the casting diagram consists of bindings which contain implicit semantics for the traceability link (i.e., between elements of the architecture and roles within the applied patterns). However, we lose traceability when creating new elements after the integration process. This is the case for security mechanisms. We have used the concepts of "Plays" for bindings. We propose to use a new concept called "Create" that links added elements in the architecture and their origin from the pattern. For instance, when integrating an SSL pattern an encryption mechanism is added in the target architecture. Hence, a link stereotyped with "Create" between this mechanism in the architecture and the SSL patten can be created. This issue of traceability has been tackled in [93]. The authors presented a traceability approach during model composition operation. Using this approach composed models contain two categories of elements: (1) elements that originate from an input model (security mechanisms in our case) and (2) elements that are the result of a merge of elements of different input models (components linked with "Plays" links in our case). Hence, they have defined two categories of traceability links: (1) Translation links and (2) Merge links. Thus, the approach reflects what have said about the need of "Create" traceability links.

2. *Interplay of concerns.* We plan to do research about safety concerns and study their interplay with security concerns. The interplay of Security and Dependability was started in the team in [62] but in a semi-formal way.

3. *Implementation Phase.* Our approach focused on the architecture design phases. We

need to inspect a way to move to the implementation phase without compromising security enforced at prior phases. For instance, Delange et al. [38] introduced an approach based on AADL [137] where the aim is to build secure applications from specifications to implementation using code generation techniques.

4. *Governance Risk and Compliance.* In this PhD thesis, we have focused on designing secure software architectures through security risk management, patterns and analyzes. However, once a software system is implemented and deployed, risks need to be managed. These risks should be linked to an organization's strategy. Governance, Risk and Compliance (GRC) refers to an organization's strategy for managing the broad issues of corporate governance, enterprise risk management (ERM) and corporate compliance with regards to regulations. For instance, Identity and Access Management (IAM) is a kind of governance about managing the life cycle of identities inside an organization (from recruitment to departure) and their impact on the information system.

Appendices

Appendix A

Security Pattern Description

This appendix describes the patterns used along the manuscript taken from [48]. We have simplified the pattern description in order to have three sections: context, problem and solution.

A.1 Transport Layer Security (TLS)

The Transport Layer Security (TLS) pattern is the latest version of Secure Socket Layer (SSL). It describes how to provide secure channel between a client and a server by which application messages are communicated over the transport layer of the Internet. The client and the server are mutually authenticated and the integrity of their data is preserved.

Context

Users using applications that exchange sensitive information such as browsers for e-commerce or similar activities. The transport layer in TCP/IP provides end-to-end communication services for applications within a layered architecture of network components and protocols, and specifically convenient services such as connection-oriented data stream support, flow control and multiplexing.

Problem

The messages communicated between applications and servers on the transport layer are vulnerable to attack by intruders, who may try to read or modify them. Either the server or the client may be an impostor.

Solution

Establish a cryptographic secure channel between the client using algorithms that can be negotiated between the client and the server. Provide the means for client and server to authenticate each other. Provide a way to preserve the integrity of messages.

Structure

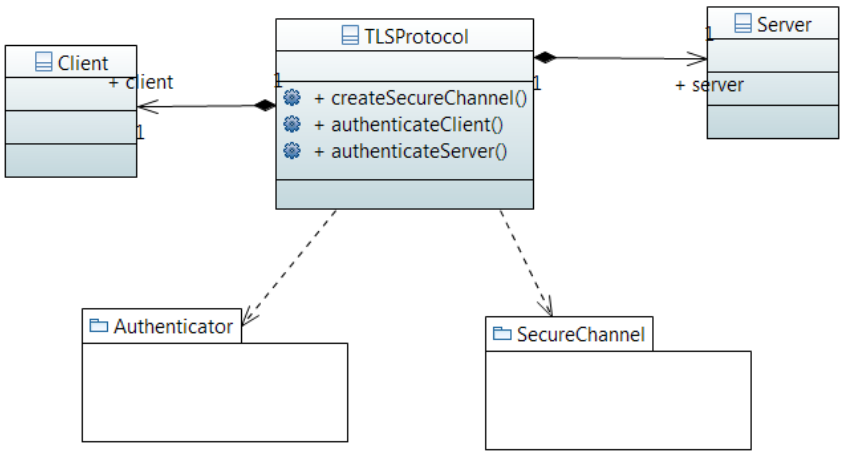


Figure A.1: Structure of TLS pattern

Figure A.1 shows a class diagram for the basic architecture of the TRANSPORT LAYER SECURITY pattern. A **Client** requests some **Service** from the **Server**. The **TLSProtocol** controller conveys this request using an **Authenticator** to mutually authenticate the **Client**, and creates a Secure Channel between them. AUTHENTICATOR and Secure Channel are patterns.

Behavior

We describe the behavior of the transport layer security pattern using a sequence diagram for the following use case:

Use Case: Request a Service – Figure A.2

- Summary: A client requests a service and the TLSProtocol authenticates the request and creates a secure channel.
- Actors: Client, Server
- Precondition: The security parameters of the secure exchange have been predefined.

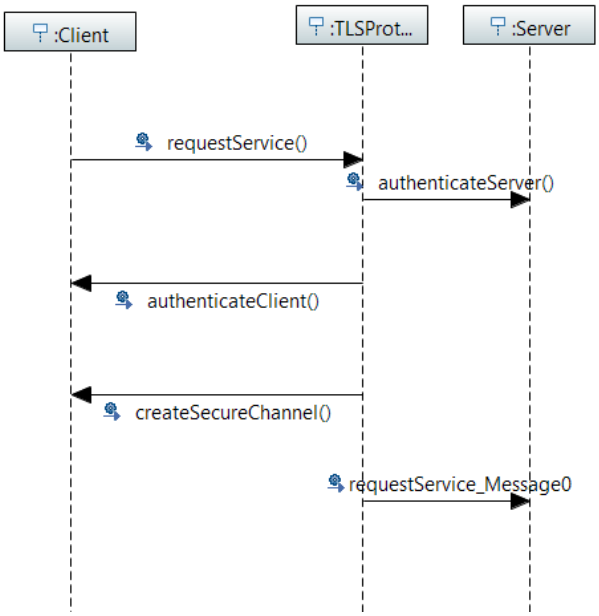


Figure A.2: Behavior of TLS pattern

- Description:
  1. The Client makes a service request to the **Server**
  2. The **TLSProtocol** authenticates the **Server** to the **Client** and to the **Server**.
  3. The TLSProtocol creates a secure channel between the **Server** and the **Client**.
- Alternate Flows:
  - The authentication can fail
  - The creation of a secure channel can fail.
- Postconditions: The **Server** accepts the request and grants the service.

A.2 Firewall

The Firewall pattern allows filtering of calls and responds to/from enterprise application, based on institution's access control policies.

Context

Enterprise applications executing in distributed systems accessed from a local network, the Internet, or other external networks. These distributed systems typically include

packet filter and/or proxy-based firewall.

Problem

Enterprise application in an organization's internal network are accessed by a broad spectrum of users that may attempt to abuse its resource (leakage, modification or destruction of data). These applications can be numerous, and thus implement access control independently in ad-hoc ways, making the system more complex and thus less secure. Moreover, traditional network firewalls (application layer firewalls or packet filters), do not make it possible to define high-level rules (role-based or individual-based rules) that could make the implementation of security policies easier and simpler. How can we control the hostile actions of users who access our application ?

Solution

Interpose a firewall that can analyze incoming requests for application services and check them for authorization. A client (user, role) can access a service of an application only if a specific policy authorizes it to do so.

A.3 Intrusion Detection System (IDS)

The IDS pattern monitors traffic as it passes through a network. It analyzes and its analysis

Context

Nodes of a local system need to communicate through an insecure network such as Internet.

Problem

An attacker may try to infiltrate the system and disrupt information integrity and confidentiality.

Solution

Each request to access the network is checked with regards to the definition of an attack. In an attack is detected, alerts are triggered and countermeasures are taken. The IDS

patterns can be realized by two concrete IDSs that operate based on the attack signature ou based abnormal behavior in the network: Signature-Based IDS or Behavior-Based IDS.

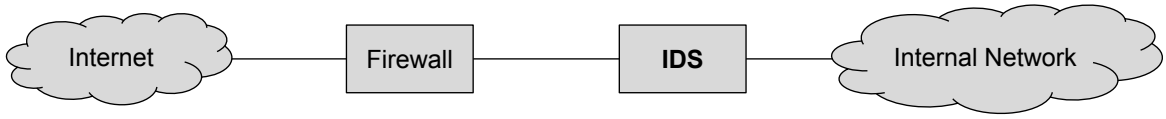


Figure A.3: Possible placement of an IDS in a network

Figure A.3 shows the basic placement of an IDS in a network as it complements a firewall. The Firewall filters the traffic and IDS checks suspicious pattern in request sequences. In this case the network operated is alerted which tells the firewall to block some or all traffic.

A.4 Logger and Auditor

The Logger and Auditor patterns keeps track of user actions when performing sensitive actions logging the identity and the time of the actions. This is useful for audit purposes.

Context

An environment with sensitive information where access to data needs to be controlled by keeping track of users actions.

Problem

How can we keep track of users actions in order to determine the responsible identity and the time of the action.

Solution

Each time a user accesses some object we record this access, indicating the user identifier, the type of access, the object accessed and the time when the access happened. The access database must be protected with authentication, authorization and encryption capabilities.

A.5 Authorization

The Authorization pattern, also known as the access matrix, describes who is authorized to access a resources in a system.

Context

A computing environment where resources have value.

Problem

Access to resources needs to be controlled, otherwise any entity (user or process) may access any resource. This may lead to information tampering and information disclosure. Hence, how to describe who is authorized to access a certain resource.

Solution

The solution is to indicate for each subject what resource it access and access type. Figure A.4, shows the different entities involved in the Authorization pattern. This pattern

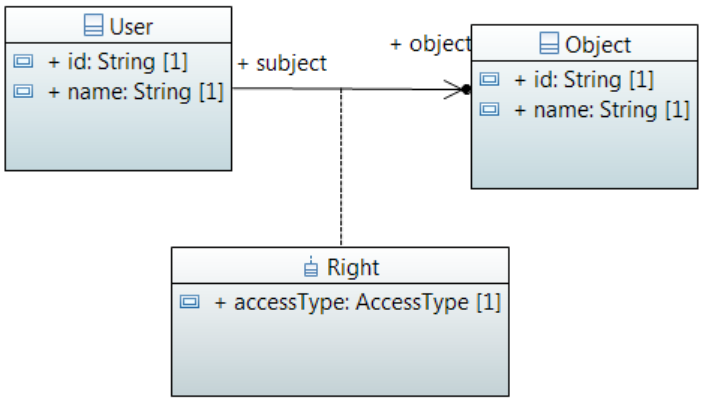


Figure A.4: Authorization Pattern Structure

is realized by two concrete Authorization patterns that are kept with resources to indicate who is authorized to access them or with processes to define their execution rights: Access Control Lists (ACLs) or Capabilities.

A.6 Role-Based Access Control (RBAC)

The Role-Based Access Control Pattern describes how to assign rights based on the functions or tasks of users in which control of access to computing resources is required.

Context

An environment where access control is needed and in which there is a large number of users or resources.

Problem

A way of factoring access rights is needed. Otherwise the number of rights would be very large, granting rights to individual users would require storing many authorization rules; and the administration of rights would be difficult. Hence, how can we reduce the number of rights and the make their semantics clearer ?

Solution

The solution is to assign rights to users based on their job function and their tasks. Job functions can be seen as roles that users play while perfoming their duties. For example, in the context of a web-based systems, roles are: company employee, employees, customers, partners, search engines, etc. Hence, each role would be assigned a set of rights and any user playing this role would automatically have these rights. Figure A.5 shows the different entities involved in the RBAC pattern.

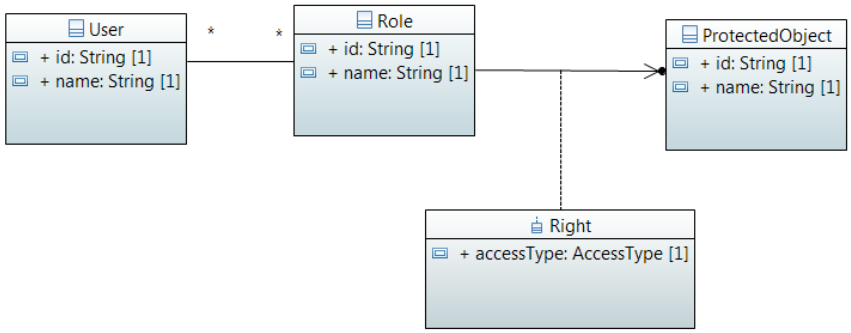


Figure A.5: RBAC Pattern Structure

Appendix B

Extracts from the threat scenarios formalized in OCL

B.1 Iteration 1

B.1.1 Man-In-The-Middle version 1

```
1  Context Application inv Man-In-The-Middle_v1
2
3  self.components->select (c1 |
4  self.components->exists (c2 |
5  not(c1.oclIsKindOf (PatternProfile::SecurityMechanism)) and
6  not(c2.oclIsKindOf (PatternProfile::SecurityMechanism)) and c1._'<>'(c2) and
7  (c1.node)._'<>'(c2.node) and
8  c1.node.channels->exists(ch | c2.node.channels->includes(ch)) and (c1.ports->exists(inp|
       c2.ports->exists(inpt2 | inpt2.communication = inp.communication)))
9  and (self.components->exists(enc | self.components->exists(mac1,mac2 | self.components->
       exists(dec |  mac1.oclIsKindOf(PatternProfile::Signer)
10 and mac2.oclIsKindOf(PatternProfile::Verifier)
11 and dec.oclIsKindOf(PatternProfile::Encryptor)
12 and  enc.oclIsKindOf(PatternProfile::Decryptor)
13 and
14   (mac1.node = c1.node)
15 and
16   (mac2.node = c2.node)
17 and
18   (enc.node = c1.node)
19 and
20   (dec.node = c2.node)
21 ))))))->size()
```

Listing B.1: Man-In-The-Middle threat scenario formalized in OCL

B.1.2 Tampering version 1

```
1  Context Application inv Tampering_v1
2
3  self.components->select(c1 |
4  self.components->exists(c2 |
5  not(c1.oclIsKindOf(PatternProfile::SecurityMechanism))
6  and
7  not(c2.oclIsKindOf(PatternProfile::SecurityMechanism))
8  and
9  c1._'<>'(c2)
10 and
11 (c1.node)._'<>'(c2.node)
12 and
13 c1.node.channels->exists(ch | c2.node.channels->includes(ch))
14 and
15 (c1.ports->exists(inp| c2.ports->exists(inpt2 | inpt2.communication = inp.communication))
       )
16 and
17 (self.components->exists(mac1,mac2 |
18   mac1.oclIsKindOf(PatternProfile::Signer)
19 and
20   mac2.oclIsKindOf(PatternProfile::Verifier)
21 and
22   (mac1.node = c1.node)
23 and
24   (mac2.node = c2.node)))))->size()
```

Listing B.2: Tampering threat scenario formalized using OCL

B.2 Iteration 2

B.2.1 Man-In-The-Middle version 2

```
1  Context Application inv Man-In-The-Middle_v2
2  self.components->select(c1 |
3  (c1.node.trustlevel=TrustLevel::untrusted or
4  c1.node.trustlevel=TrustLevel::untrusted)
5
6  and
7  self.components->size(c2 |
8  not(c1.oclIsKindOf(PatternProfile::SecurityMechanism))
9  and
10 not(c2.oclIsKindOf(PatternProfile::SecurityMechanism))
11 and
12 c1._'<>'(c2)
13 and
14 (c1.node)._'<>'(c2.node)
15 and
16
```

```
17 c1.node.channels->exists(ch | c2.node.channels->includes(ch))
18 and
19 --does c1 and c2 communicate ?
20 (c1.ports->exists(inp| c2.ports->exists(inpt2 | inpt2.communication = inp.communication))
       )
21  and
22 --the security components are correctly used
23 (self.components->select(enc |  self.components->exists(dec , mac1 | self.components->
       exists(mac2| enc.oclIsKindOf(PatternProfile::Encryptor)
24 and
25  dec.oclIsKindOf(PatternProfile::Decryptor)
26 and
27  mac1.oclIsKindOf(PatternProfile::Signer)
28 and
29  mac2.oclIsKindOf(PatternProfile::Verifier)
30 and
31  (enc.node = c1.node)
32 and
33   (dec.node = c2.node)
34 and
35   (mac1.node = c1.node)
36 and
37   (mac2.node = c2.node)
38 and
39 (
40 -- Mecanisms are connected to components
41
42 (c1.ports->exists(inp_c1 |  enc.ports->exists(inp_enc | inp_c1.communication = inp_enc.
       communication and inp_enc.communication._'<>'(null)))
43      and
44  c1.ports->exists(inp_enc |  mac1.ports->exists(inp_mac1 | inp_mac1.communication =
       inp_enc.communication and inp_enc.communication._'<>'(null)))
45  and
46  c2.ports->exists(inp_mac1 |  mac2.ports->exists(inp_mac2 | inp_mac2.communication =
       inp_mac1.communication and inp_mac1.communication._'<>'(null)))
47  and
48  c2.ports->exists(inp_mac2 |  dec.ports->exists(inp_dec | inp_mac2.communication =
       inp_dec.communication and  inp_dec.communication._'<>'(null)))
49  and
50
51 -- the mechanisms are called correctly and secret data goes through 'encryptor' and '
       signer' before to the other component
52 c1.ports->select(inp_c1_c2 | inp_c1_c2.msg_out._'<>'(null) and (c1.ports->exists(inp| c2.
       ports->exists(inpt2 | inpt2.communication = inp_c1_c2.communication))))->forAll(
       inp_c1_c2 | mac1.ports->exists(sign_in |c1.ports->exists(c1_inp| c1_inp.communication
       = sign_in.communication  )
53  and
54  (enc.ports->exists(enc_in | c1.ports->exists(c1_signorEnc |
55 --message flow scenario 1
56       (c1_signorEnc.communication = enc_in.communication
57       and
58       sign_in.msg_out = inp_c1_c2.msg_out
59       and
60       sign_in.msg_in = enc_in.msg_out
```

```
61      and
62      enc_in.msg_in = c1_signorEnc.msg_out)
63           or
64 --message flow scenario 2
65      (c1_signorEnc.communication = sign_in.communication
66      and
67      enc_in.msg_out = inp_c1_c2.msg_out
68      and
69      enc_in.msg_in = sign_in.msg_out
70      and
71      sign_in.msg_in = c1_signorEnc.msg_out))))))))))->isEmpty()))->size()
```

Listing B.3: Man-In-The-Middle threat scenario version 2 formalized in OCL

B.2.2 Tampering version 2

```
1  Context Application inv Tampering_v2
2
3  self.components->select(c1 |
4  self.components->exists(c2 |
5  not(c1.oclIsKindOf(PatternProfile::SecurityMechanism))
6  and
7  not(c2.oclIsKindOf(PatternProfile::SecurityMechanism))
8  and
9  c1._'<>'(c2)
10 and
11 (c1.node)._'<>'(c2.node)
12 and
13
14 c1.node.channels->exists(ch | c2.node.channels->includes(ch))
15 and
16 --does c1 and c2 communicate ?
17 (c1.ports->exists(inp| c2.ports->exists(inpt2 | inpt2.communication = inp.communication))
       )
18  and
19 --the security components are correctly used
20  (
21
22 self.components->select(enc |
23  self.components->exists(dec , mac1 |
24  self.components->exists(mac2|
25   enc.oclIsKindOf(PatternProfile::Encryptor)
26  and
27    dec.oclIsKindOf(PatternProfile::Decryptor)
28  and
29   mac1.oclIsKindOf(PatternProfile::Signer)
30   and
31   mac2.oclIsKindOf(PatternProfile::Verifier)
32  and
33   (enc.node = c1.node)
34  and
35   (dec.node = c2.node)
```

```
36 and
37   (mac1.node = c1.node)
38     and
39   (mac2.node = c2.node)
40  and
41   mac1._'<>'(mac2)
42  -- and
43  -- (mac1.oclAsType(Signer).usedKeyKind)._'<>'(KeyKind::None)
44  -- and
45  -- (mac2.oclAsType(Verifier).usedKeyKind)._'<>'(KeyKind::None)
46
47  and
48 (mac2.oclAsType(Verifier).usedKeyKind) = mac1.oclAsType(Signer).usedKeyKind
49
50     and
51
52
53    (
54        --Encryption followed by HMAC e.g., IPsec
55        -- Mecanisms are connected to components
56    (c1.ports->exists(inp_c1 |  enc.ports->exists(inp_enc | inp_c1.communication = inp_enc
          .communication and inp_enc.communication._'<>'(null)))
57  and
58
59    c1.ports->exists(inp_enc |  mac1.ports->exists(inp_mac1 | inp_mac1.communication =
          inp_enc.communication and inp_enc.communication._'<>'(null)))
60  and
61    c2.ports->exists(inp_mac1 |  mac2.ports->exists(inp_mac2 | inp_mac2.communication =
          inp_mac1.communication and inp_mac1.communication._'<>'(null)))
62     and
63    c2.ports->exists(inp_mac2 |  dec.ports->exists(inp_dec | inp_mac2.communication =
          inp_dec.communication and  inp_dec.communication._'<>'(null)))
64    and
65    -- the mechanisms are called correctly and secret data goes through 'encryptot' and '
          signer' before to the other component
66    c1.ports->select(inp_c1_c2 | inp_c1_c2.msg_out._'<>'(null) and (c1.ports->exists(inp|
          c2.ports->exists(inpt2 | inpt2.communication = inp_c1_c2.communication))))->forAll
          (inp_c1_c2 |
67     mac1.ports->exists(sign_in |
68      c1.ports->exists(c1_inp| c1_inp.communication = sign_in.communication  )
69      and
70      (enc.ports->exists(enc_in |
71       c1.ports->exists(c1_signorEnc |
72      --data flow scenario 1
73      (c1_signorEnc.communication = enc_in.communication
74      and
75      sign_in.msg_out = inp_c1_c2.msg_out
76      and
77      sign_in.msg_in = enc_in.msg_out
78      and
79      enc_in.msg_in = c1_signorEnc.msg_out)
80           or
81
82      --data flow scenario 2
```

```
83        (c1_signorEnc.communication = sign_in.communication
84        and
85        enc_in.msg_out = inp_c1_c2.msg_out
86        and
87        enc_in.msg_in = sign_in.msg_out
88        and
89        sign_in.msg_in = c1_signorEnc.msg_out)
90
91        ))
92 ))))
93     )
94
95    )
96  ))->isEmpty()
97
98  and
99  -- No Message protection functions set of mac1 and mac2 linked to c1 and c2 is empty
100  self.components->select(signer |
101   signer.oclIsKindOf(Signer)
102   and
103   signer.node = c1.node
104   and
105   self.components->exists(verifier |
106    verifier.oclIsKindOf(Verifier)
107        and
108   verifier.node = c2.node
109
110  and
111 (verifier.oclAsType(Verifier).usedKeyKind) =signer.oclAsType(Signer).usedKeyKind
112    and
113   c1.ports->exists(inp_mac1 |  signer.ports->exists(inp_mac2 | inp_mac2.communication =
        inp_mac1.communication and inp_mac1.communication._'<>'(null)))
114    and
115   c2.ports->exists(inp_dec |  verifier.ports->exists(inp_c2 | inp_c2.communication =
        inp_dec.communication and  inp_dec.communication._'<>'(null))))
116
117   and
118   -- the mechanisms are called correctly and secret data goes through 'encryptot' and '
        signer' before to the other component
119   c1.ports->select(inp_c1_c2 | inp_c1_c2.msg_out._'<>'(null) and (c1.ports->exists(inp|
        c2.ports->exists(inpt2 | inpt2.communication = inp_c1_c2.communication))))->forAll
        (inp_c1_c2 |
120    signer.ports->exists(sign_in |
121     c1.ports->exists(c1_inp| c1_inp.communication = sign_in.communication  )
122     and
123
124      c1.ports->exists(c1_sign |
125
126     --data flow scenario 3
127     (c1_sign.communication = sign_in.communication
128     and
129     sign_in.msg_out = inp_c1_c2.msg_out
130     and
131     c1_sign.msg_out = sign_in.msg_in
```

```
132  )))))->isEmpty()
133  )
134 ))->size()
```

Listing B.4: Tampering threat scenario version 2 formalized using OCL

B.3 Iteration 3

In this iteration, we consider Denial-of-Service and Injection threat scenarios.

B.3.1 Denial of Service version 1

```
1  Context Application inv DenialofService_v1
2  self.components->select(c1 |
3  c1.ports->exists(port |
4  self.components->select(firewall |
5  --any kind of firewall
6  firewall.oclIsKindOf(Firewall)
7  and
8  firewall.node = c1.node
9  and
10 self.components->exists(auth |
11 auth.oclIsKindOf(Authenticator)
12 and
13 auth.node = c1.node
14 and
15 self.components->exists(authorizer |
16 authorizer.oclIsKindOf(Authorizer)
17 and
18 authorizer.node = c1.node
19 and
20 c1.ports->exists(in_c1_firewall |  firewall.ports->exists(inp_firewall |
21 in_c1_firewall.communication = in_c1_firewall.communication
22 and
23 in_c1_firewall.communication._'<>'(null)
24 and
25 port.msg_in = in_c1_firewall.msg_out
26 and
27 in_c1_firewall.msg_out = inp_firewall.msg_in
28 and
29 in_c1_firewall.msg_in = inp_firewall.msg_out
30 and
31 c1.ports->exists(in_c1_auth |  auth.ports->exists(inp_auth|
32 in_c1_auth.communication = inp_auth.communication
33 and
34 in_c1_auth.communication._'<>'(null)
35 and
36 in_c1_auth.msg_out =  in_c1_firewall.msg_in
37 and
38 inp_auth.msg_in =  in_c1_auth.msg_out
```

```
39 and
40 inp_auth.msg_out =  in_c1_auth.msg_in
41 and
42 c1.ports->exists(in_c1_authorizer |  authorizer.ports->exists(inp_authorizer|
43 inp_authorizer.communication = in_c1_authorizer.communication
44 and
45 in_c1_authorizer.communication._'<>'(null)
46 and
47 in_c1_authorizer.msg_out =  in_c1_auth.msg_in
48 and
49 inp_authorizer.msg_in =  in_c1_authorizer.msg_out
50 and
51 inp_authorizer.msg_out =  in_c1_authorizer.msg_in )))))))))->isEmpty()))->size()
```

Listing B.5: Denial of Service threat scenario formalized using OCL

B.3.2 Injection threat version 1

```
1  Context Application inv Injection_v1
2  self.components->select(c1 |
3  c1.ports->exists( port |
4  self.components->select(firewall |
5  --any kind of firewall
6  firewall.oclIsKindOf(Firewall)
7  and
8  firewall.node = c1.node
9  and
10 self.components->exists(auth |
11 auth.oclIsKindOf(Authenticator)
12 and
13 auth.node = c1.node
14 and
15 self.components->exists(authorizer |
16 authorizer.oclIsKindOf(Authorizer)
17 and
18 authorizer.node = c1.node
19 and
20 c1.ports->exists(in_c1_firewall |  firewall.ports->exists(inp_firewall |
21 in_c1_firewall.communication = in_c1_firewall.communication
22 and
23 in_c1_firewall.communication._'<>'(null)
24 and
25 port.msg_in = in_c1_firewall.msg_out
26 and
27 in_c1_firewall.msg_out = inp_firewall.msg_in
28 and
29 in_c1_firewall.msg_in = inp_firewall.msg_out
30 and
31 c1.ports->exists(in_c1_auth |  auth.ports->exists(inp_auth|
32 in_c1_auth.communication = inp_auth.communication
33 and
34 in_c1_auth.communication._'<>'(null)
```

```
35 and
36 in_c1_auth.msg_out =  in_c1_firewall.msg_in
37 and
38 inp_auth.msg_in =  in_c1_auth.msg_out
39 and
40 inp_auth.msg_out =  in_c1_auth.msg_in
41 and
42 c1.ports->exists(in_c1_authorizer |  authorizer.ports->exists(inp_authorizer|
43 inp_authorizer.communication = in_c1_authorizer.communication
44 and
45 in_c1_authorizer.communication._'<>'(null)
46 and
47 in_c1_authorizer.msg_out =  in_c1_auth.msg_in
48 and
49 inp_authorizer.msg_in =  in_c1_authorizer.msg_out
50 and
51 inp_authorizer.msg_out =  in_c1_authorizer.msg_in))))))))->isEmpty()))->size()
```

Listing B.6: Injection threat scenario formalized using OCL

B.4 Iteration 4

In this iteration, we add the concept of port types i.e., if the port is public or private.

B.4.1 Denial of Service version 2

```
1 Context Application inv DenialofService_v1
2 self.components->select(c1 |
3 c1.ports->exists(public_port |
4 (public_port.portkind = portKind::external)
5 and
6 self.components->select(firewall |
7 --any kind of firewall
8 firewall.oclIsKindOf(Firewall)
9 and
10 firewall.node = c1.node
11 and
12 self.components->exists(auth |
13 auth.oclIsKindOf(Authenticator)
14 and
15 auth.node = c1.node
16 and
17 self.components->exists(authorizer |
18 authorizer.oclIsKindOf(Authorizer)
19 and
20 authorizer.node = c1.node
21 and
22 c1.ports->exists(in_c1_firewall |  firewall.ports->exists(inp_firewall |
23 in_c1_firewall.communication = in_c1_firewall.communication
24 and
```

```
25 in_c1_firewall.communication._'<>'(null)
26 and
27 public_port.msg_in = in_c1_firewall.msg_out
28 and
29 in_c1_firewall.msg_out = inp_firewall.msg_in
30 and
31 in_c1_firewall.msg_in = inp_firewall.msg_out
32 and
33 c1.ports->exists(in_c1_auth |  auth.ports->exists(inp_auth|
34 in_c1_auth.communication = inp_auth.communication
35 and
36 in_c1_auth.communication._'<>'(null)
37 and
38 in_c1_auth.msg_out =  in_c1_firewall.msg_in
39 and
40 inp_auth.msg_in =  in_c1_auth.msg_out
41 and
42 inp_auth.msg_out =  in_c1_auth.msg_in
43 and
44 c1.ports->exists(in_c1_authorizer |  authorizer.ports->exists(inp_authorizer|
45 inp_authorizer.communication = in_c1_authorizer.communication
46 and
47 in_c1_authorizer.communication._'<>'(null)
48 and
49 in_c1_authorizer.msg_out =  in_c1_auth.msg_in
50 and
51 inp_authorizer.msg_in =  in_c1_authorizer.msg_out
52 and
53 inp_authorizer.msg_out =  in_c1_authorizer.msg_in )))))))))->isEmpty()))->size()
```

Listing B.7: Denial of Service threat scenario formalized using OCL

B.4.2 Injection threat version 2

```
1 Context Application inv Injection_v1
2 self.components->select(c1 |
3 c1.ports->exists( public_port |
4 (public_port.Portkind = PortKind::external)
5 and
6 self.components->select(firewall |
7 --any kind of firewall
8 firewall.oclIsKindOf(Firewall)
9 and
10 firewall.node = c1.node
11 and
12 self.components->exists(auth |
13 auth.oclIsKindOf(Authenticator)
14 and
15 auth.node = c1.node
16 and
17 self.components->exists(authorizer |
18 authorizer.oclIsKindOf(Authorizer)
```

```
19 and
20 authorizer.node = c1.node
21 and
22 portputs->exists(in_c1_firewall |  firewall.ports->exists(inp_firewall |
23 in_c1_firewall.communication = in_c1_firewall.communication
24 and
25 in_c1_firewall.communication._'<>'(null)
26 and
27 public_port.msg_in = in_c1_firewall.msg_out
28 and
29 in_c1_firewall.msg_out = inp_firewall.msg_in
30 and
31 in_c1_firewall.msg_in = inp_firewall.msg_out
32 and
33 c1.ports->exists(in_c1_auth |  auth.ports->exists(inp_auth|
34 in_c1_auth.communication = inp_auth.communication
35 and
36 in_c1_auth.communication._'<>'(null)
37 and
38 in_c1_auth.msg_out =  in_c1_firewall.msg_in
39 and
40 inp_auth.msg_in =  in_c1_auth.msg_out
41 and
42 inp_auth.msg_out =  in_c1_auth.msg_in
43 and
44 c1.ports->exists(in_c1_authorizer |  authorizer.ports->exists(inp_authorizer|
45 inp_authorizer.communication = in_c1_authorizer.communication
46 and
47 in_c1_authorizer.communication._'<>'(null)
48 and
49 in_c1_authorizer.msg_out =  in_c1_auth.msg_in
50 and
51 inp_authorizer.msg_in =  in_c1_authorizer.msg_out
52 and
53 inp_authorizer.msg_out =  in_c1_authorizer.msg_in))))))))->isEmpty()))->size()
```

Listing B.8: Injection threat scenario formalized using OCL

Bibliography

[1] CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC). `http://capec.mitre.org/`. [Accessed: May-2016].

[2] CWE - Common Weakness Enumeration. `https://cwe.mitre.org/`. [Accessed: April-2016].

[3] IEEE Standard Communication Delivery Time Performance Requirements for Electric Power Substation Automation. *IEEE Std 1646-2004*, pages 0_1–24, 2005.

[4] European Commission : CORDIS : Projects & Results Service : System engineering for security and dependability. `http://cordis.europa.eu/project/rcn/78381_en.html`, 2006. [Accessed: April-2016].

[5] Systems and software engineering Vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, pages 1–418, 2010.

[6] Attackers Alter Water Treatment Systems in Utility Hack: Report | SecurityWeek.Com. `http://www.securityweek.com/attackers-alter-water-treatment-systems-utility-hack-report`, 2015. [Accessed: December-2016].

[7] R. Abdallah, A. Motii, N. Yakymets, and A. Lanusse. Using Model Driven Engineering to Support Multi-paradigms Security Analysis. In *Model-Driven Engineering and Software Development - Third International Conference*, volume 580 of *MODELSWARD '15*, pages 278–292. Springer, 2015.

[8] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[9] A. Alkussayer and W. H. Allen. A scenario-based framework for the security evaluation of software architecture. In *2010 3rd IEEE International Conference on*

Computer Science and Information Technology (ICCSIT), volume 5, pages 687–695, 2010.

[10] M. Almorsy, J. Grundy, and A. S. Ibrahim. Automated Software Architecture Security Risk Analysis Using Formalized Signatures. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 662–671. IEEE Press, 2013.

[11] A. Alshamsi and T. Saito. A technical comparison of IPSec and SSL. In *19th International Conference on Advanced Information Networking and Applications*, volume 2 of *AINA '05*, pages 395–398 vol.2, 2005.

[12] A. Alvi and M. Zulkernine. A Comparative Study of Software Security Pattern Classifications. In *2012 Seventh International Conference on Availability, Reliability and Security*, ARES'12, pages 582–589, 2012.

[13] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu. Evaluating Goal Models Within the Goal-oriented Requirement Language. *Int. J. Intell. Syst.*, 25(8):841–877, 2010.

[14] ANSSI. EBIOS 2010: Expression des besoins et Identification des Objectifs de Sécurité (2010). `http://www.ssi.gouv.fr/`, 2010. [Accessed: May-2014].

[15] P. Antonino, S. Duszynski, C. Jung, and M. Rudolph. Indicator-based Architecture-level Security Evaluation in a Service-oriented Environment. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, pages 221–228. ACM, 2010.

[16] C. Atkinson, J. Bayer, and D. Muthig. Component-Based Product Line Development: The KobrA Approach. In *Software Product Lines*, The Springer International Series in Engineering and Computer Science, pages 289–309. Springer, Boston, MA, 2000.

[17] H. Attiya and J. Welch. Basic Algorithms in Message-Passing Systems. In *Distributed Computing*, pages 7–29. John Wiley & Sons, Inc., 2004.

[18] A. Bagnato and Bagnato. *Handbook of Research on Embedded Systems Design*. IGI Global, 2014.

[19] D. Basin, M. Clavel, J. Doser, and M. Egea. Automated analysis of security-design models. *Information and Software Technology*, (5):815, 2009.

[20] I. Bayley and H. Zhu. Formalising design patterns in predicate logic. In *Software Engineering and Formal Methods, 2007. SEFM 2007. Fifth IEEE International Conference on*, pages 25–36. IEEE, 2007.

[21] K. Beck and R. Johnson. *Patterns generate architectures*, pages 139–149. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[22] B. J. Berger, K. Sohr, and R. Koschke. Extracting and Analyzing the Implemented Security Architecture of Business Applications. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 285–294. IEEE, 2013.

[23] S. Bernardi, J. Merseguer, and D. C. Petriu. A dependability profile within MARTE. *Software and System Modeling*, 10(3):313–336, 2011.

[24] S. Bernardi, J. Merseguer, and D. C. Petriu. Dependability modeling and analysis of software systems specified with UML. *ACM Comput. Surv.*, 45(1):2, 2012.

[25] J. Bézivin. Towards a precise definition of the omg/mda framework. In *Proceedings of ASE'01*, pages 273–280. IEEE Computer Society Press, 2001.

[26] M. Borek, N. Moebius, K. Stenzel, and W. Reif. Security requirements formalized with OCL in a model-driven approach. In *2013 3rd International Workshop on Model-Driven Requirements Engineering (MoDRE)*, pages 65–73, July 2013.

[27] F. Braz, E. Fernandez, and M. VanHilst. Eliciting Security Requirements through Misuse Activities. In *19th International Workshop on Database and Expert Systems Application, 2008. DEXA '08*, pages 328–333, Sept. 2008.

[28] M. Bunke, R. Koschke, and K. Sohr. Organizing security patterns related to security and pattern recognition requirements. *International Journal on Advances in Security*, 5:46–67, 2012.

[29] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Son Ltd, 1996.

[30] A. Calder. *Information Security Based on ISO 27001/ISO 27002: A Management Guide - Best Practice*. Van Haren Publishing, 2009.

[31] R. Caralli, J. Stevens, L. Young, and W. Wilson. Introducing OCTAVE Allegro: Improving the Information Security Risk Assessment Process. `http://resources.`

sei.cmu.edu/library/asset-view.cfm?AssetID=8419, 2007. [Accessed: June-2014].

[32] R. S. Chakraborty and S. Bhunia. Security through obscurity: An approach for protecting Register Transfer Level hardware IP. In *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 96–99. IEEE, 2009.

[33] V. Cortellessa, C. Trubiani, L. Mostarda, and N. Dulay. An Architectural Framework for Analyzing Tradeoffs between Software Security and Performance. In *Architecting Critical Systems*, number 6150 in Lecture Notes in Computer Science, pages 1–18. Springer Berlin Heidelberg, 2010.

[34] I. Crnkovic. *Building Reliable Component-Based Software Systems*. Artech House, Inc., Norwood, MA, USA, 2002.

[35] A. Daneels and W. Salter. What is SCADA? In *International Conference on Accelerator and Large Experimental Physics Control Systems*, ICALEPCS'99, 1999.

[36] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(35):35:1–35:44, 2011.

[37] V. Debruyne, F. Simonot-Lion, and Y. Trinquet. EAST-ADL - An Architecture Description Language. In *Architecture Description Languages*, IFIP The International Federation for Information Processing, pages 181–195. Springer US, 2005.

[38] J. Delange, L. Pautet, and F. Kordon. Design, implementation and verification of MILS systems. *Softw., Pract. Exper.*, 42(7):799–816, 2012.

[39] F. den Braber, I. Hogganvik, M. S. Lund, K. Stølen, and F. Vraalsen. Model-based security analysis in seven steps — a guided tour to the coras method. *BT Technology Journal*, 25(1):101–117, 2007.

[40] J. Dong, T. Peng, and Y. Zhao. Automated verification of security pattern compositions. *Information and Software Technology*, 52(3):274–295, 2010.

[41] D. F. D'Souza and A. C. Wills. *Objects, components, and frameworks with UML : the catalysis approach*. Addison-Wesley Professional, 1998.

[42] E. Dubois, P. Heymans, N. Mayer, and R. Matulevičius. A Systematic Approach to Define the Domain of Information System Security Risk Management. In *Intentional*

Perspectives on Information Systems Engineering, pages 289–306. Springer Berlin Heidelberg, 2010.

[43] Eclipse Foundantion. CDO Model Repository Overview. `http://www.eclipse.org/cdo/`, 2013. [Accessed: August-2017].

[44] Eclipse Foundation. Qvt operational language. https://projects.eclipse.org/projects/modeling.mmt.qvt-oml, 2013.

[45] P. El Khoury, A. Mokhtari, E. Coquery, and M.-S. Hacid. An Ontological Interface for Software Developers to Select Security Patterns. In *19th International Workshop on Database and Expert Systems Application, 2008. DEXA '08*, pages 297–301, Sept. 2008.

[46] H. Espinoza, D. Servat, and S. Gérard. Leveraging analysis-aided design decision knowledge in UML-based development of embedded systems. In *Proceedings of the 3rd International Workshop on Sharing and Reusing Architectural Knowledge*, SHARK '08, pages 55–62, 2008.

[47] E. B. Fernandez. Using security patterns to develop secure systems. In *Software engineering for secure systems. Industrial and research perspectives*, pages 16–31, 2011.

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

[49] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

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

[51] A. Fuchs, S. Gurgens, and C. Rudolph. Towards a Generic Process for Security Pattern Integration. In *20th International Workshop on Database and Expert Systems Application, 2009. DEXA '09*, pages 171–175, Aug. 2009.

[52] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and G. Booch. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition edition, 1994.

[53] G. Georg, I. Ray, K. Anastasakis, B. Bordbar, M. Toahchoodee, and S. H. Houmb. An aspect-oriented methodology for designing secure applications. *Information and Software Technology*, 51(5):846–864, May 2009.

[54] I. Graham, B. Henderson-Sellers, and H. Younessi. *The OPEN Process Specification*. ACM Press/Addison-Wesley Publishing Co., 1997.

[55] H. Grandy, D. Haneberg, W. Reif, and K. Stenzel. Developing Provable Secure M-Commerce Applications. In *Emerging Trends in Information and Communication Security*, pages 115–129. Springer, Berlin, Heidelberg, 2006.

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

[57] J. Großmann and F. Seehusen. Combining Security Risk Assessment and Security Testing Based on Standards. In *Risk Assessment and Risk-Driven Testing*, pages 18–33. Springer, 2015.

[58] D. Gross and E. Yu. From Non-Functional Requirements to Design through Patterns. *Requirements Engineering*, 6(1):18–36, 2001.

[59] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. volume 34, pages 133–153. IEEE, 2008.

[60] J. G. Hall, M. Jackson, R. C. Laney, B. Nuseibeh, and L. Rapanotti. Relating software requirements and architectures using problem frames. In *IEEE Joint International Conference on Requirements Engineering*, pages 137–144. IEEE, 2002.

[61] B. Hamid. Modeling of Secure and Dependable Applications Based on a Repository of Patterns: The SEMCO Approach. *Reliability Digest, IEEE Reliability Society, Special Issue on Trustworthy Computing and Cybersecurity*, 1(1):9–17, 2014.

[62] B. Hamid. Interplay of Security&Dependability and Resource Using Model-Driven and Pattern-Based Development. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 254–262. IEEE, 2015.

[63] B. Hamid. A Model-Driven Approach for Developing a Model Repository: Methodology and Tool Support. *Future Generation Computer Systems, Elsevier*, 68:473–490, 2017.

[64] B. Hamid, J. Geisel, A. Ziani, J. Bruel, and J. Perez. Model-driven engineering for trusted embedded systems based on security and dependability patterns. In *SDL Forum*, pages 72–90, 2013.

[65] B. Hamid, S. Gürgens, and A. Fuchs. Security patterns modeling and formalization for pattern-based development of secure software systems. *ISSE*, 12(2):109–140, 2016.

[66] B. Hamid, C. Percebois, and D. Gouteux. A Methodology for Integration of Patterns with Validation Purpose. In *Proceedings of the 17th European Conference on Pattern Languages of Programs*, EuroPLoP '12, pages 8:1–8:14. ACM, 2012.

[67] B. Hamid and J. Perez. Supporting Pattern-Based Dependability Engineering via Model-Driven Development: Approach, tool-support and empirical validation. *Journal of Systems and Software, Elsevier*, 122:239–273, 2016.

[68] M. G. Harbour, J. J. G. García, J. C. P. Gutiérrez, and J. M. D. Moyano. Mast: Modeling and analysis suite for real time applications. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, ECRTS '01, pages 125–134. IEEE Computer Society, 2001.

[69] M. G. Harbour, J. J. Gutiérrez, J. M. Drake, P. L. Martínez, and J. C. Palencia. Modeling distributed real-time systems with MAST 2. *Journal of Systems Architecture*, 59(6):331 – 340, 2013.

[70] D. Harel and B. Rumpe. Modeling languages: Syntax, semantics and all that stuff, part i: The basic stuff. `http://www.se-rwth.de/~rumpe/publications/Modeling-Languages-Syntax-Semantics-and-All-That-Stuff.pdf`, 2000. [Accessed: July-2014].

[71] D. Hartmeier. Design and Performance of the OpenBSD Stateful Packet Filter (Pf). In *Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference*, pages 171–180. USENIX Association, 2002.

[72] S. Hasheminejad and S. Jalili. Selecting Proper Security Patterns Using Text Classification. In *International Conference on Computational Intelligence and Software Engineering, 2009. CiSE 2009*, pages 1–5, 2009.

[73] S. M. H. Hasheminejad and S. Jalili. Design patterns selection: An automatic two-phase method. *Journal of Systems and Software*, 85(2):408–424, 2012.

[74] D. Hatebur, M. Heisel, and H. Schmidt. A Security Engineering Process based on Patterns. In *18th International Workshop on Database and Expert Systems Applications*, DEXA'07, pages 734–738, 2007.

[75] J. M. Horcas, M. Pinto, and L. Fuentes. An Aspect-Oriented Model transformation to weave security using CVL. In *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 138–150, Jan. 2014.

[76] S. H. Houmb, G. Georg, J. Jürjens, and R. France. An integrated security verification and security solution design trade-off analysis approach. *Integrating Security and Software Engineering: Advances and Future Visions/Mouratidis, Haralambos*, pages 190–219, 2007.

[77] IEC. Communication networks and systems in substations – Part 5: Communication requirements for functions and device models. `https://iecwebstore.com/p-preview/info_iec61850-5%7Bed1.0%7Den.pdf`, 2003. [Accessed: April-2016].

[78] ISO/IEC. Information technology – Security techniques – Information security risk management. `http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=56742`, 2011. [Accessed: April-2016].

[79] ISO/IEC. Information technology – Security techniques – Information security management systems – Requirements, 2013.

[80] ISO/IEC. Information technology – Security techniques – Information security management systems – Overview and vocabulary, 2014.

[81] M. Jan, C. Jouvray, F. Kordon, A. Kung, J. Lalande, F. Loiret, J. F. Navas, L. Pautet, J. Pulou, A. Radermacher, and L. Seinturier. Flex-eware: a flexible model driven solution for designing and implementing embedded distributed systems. *Softw., Pract. Exper.*, 42(12):1467–1494, 2012.

[82] J. Jensen and M. G. Jaatun. Security in model driven development: A survey. In *Proceedings of the 2011 Sixth International Conference on Availability, Reliability and Security. ARES '11*, pages 704–709. IEEE Computer Society, 2011.

[83] Joint Task Force Transformation Initiative. Guide for conducting risk assessments. Technical report, National Institute of Standards and Technology, 2012. [Accessed: August-2014].

[84] J. Jürjens. Towards development of secure systems using umlsec. In *Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering*, FASE '01, pages 187–200. Springer-Verlag, 2001.

[85] J. Jürjens. UMLsec: Extending UML for Secure Systems Development. In J.-M. Jézéquel, H. H. smann, and S. Cook, editors, *UML 2002 - The Unified Modeling Language, 5th International Conference, Dresden, Germany, September 30 - October 4, 2002, Proceedings*, volume 2460 of *Lecture Notes in Computer Science*, pages 412–425. Springer, 2002.

[86] R. Kazman, L. Bass, G. Abowd, and M. Webb. SAAM: A method for analyzing the properties of software architectures. In *16th International Conference on Software Engineering*, ICSE'16, pages 81–90. IEEE, 1994.

[87] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere. The architecture tradeoff analysis method. In *Fourth IEEE International Conference on Engineering of Complex Computer Systems*, ICECCS'98, pages 68–78, 1998.

[88] R. Z. Khan and J. Ali. Classification of task partitioning and load balancing strategies in distributed parallel computing systems. *International Journal of Computer Applications*, 60(17), 2012.

[89] M. Kircher and M. Völter. Guest Editors' Introduction: Software Patterns. *IEEE Software*, 24(4):28–30, 2007.

[90] A. G. Kleppe. A language description is more than a metamodel. In *Fourth International Workshop on Software Language Engineering, Nashville, USA*, page 9. megaplanet.org, 2007.

[91] A. G. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[92] R. T. Kolagari, D. Chen, A. Lanusse, R. Librino, H. Lönn, N. Mahmud, C. Mraidha, M. Reiser, S. Torchiaro, S. T. Piergiovanni, T. Wägemann, and N. Yakymets. Model-based analysis and engineering of automotive architectures with EAST-ADL: revisited. *IJCSSA*, 3(2):25–70, 2015.

[93] Y. Laghouaouta, A. Anwar, M. Nassar, and J. Bruel. A generic traceability framework for model composition operation. In *Enterprise, Business-Process and In-*

formation Systems Modeling - 16th International Conference, BPMDS'15, pages 461–475, 2015.

[94] S. Lamb. Security features in windows vista and ie7 microsoft's view. *Network Security*, 2006(8):3 – 7, 2006.

[95] Y. Lee, J. Lee, and Z. Lee. Integrating Software Lifecycle Process Standards with Security Engineering. *Computers & Security*, 21(4):345–355, 2002.

[96] Y. Lee, Z. Lee, and C. K. Lee. A study of integrating the security engineering process into the software lifecycle process standard (IEEE/EIA 12207). *AMCIS 2000 Proceedings*, page 182, 2000.

[97] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Proceedings. Real-Time Systems Symposium*, pages 166–171, 1989.

[98] P. Leserf, P. d. Saqui-Sannes, and J. Hugues. Multi domain optimization with SysML modeling. In *20th Conference on Emerging Technologies Factory Automation*, ETFA '15, pages 1–8. IEEE, 2015.

[99] T. Lodderstedt, D. A. Basin, and J. Doser. Secureuml: A uml-based modeling language for model-driven security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 426–441. Springer-Verlag, 2002.

[100] L. Lucio, Q. Zhang, P. H. Nguyen, M. Amrani, J. Klein, H. Vangheluwe, and Y. L. Traon. Advances in model-driven security. *Advances in Computers*, 93:103–152, 2014.

[101] A. Maña, E. Damiani, S. Gürgens, and G. Spanoudakis. Extensions to Pattern Formats for Cyber Physical Systems. In *Proceedings of the 31st Conference on Pattern Languages of Programs*, number 15 in PLoP'14, pages 15:1–15:8. ACM, 2014.

[102] D. Manolescu, W. Kozaczynski, A. Miller, and J. Hogg. The Growing Divide in the Patterns World. *IEEE Software*, 24(4):61–67, July 2007.

[103] D. Mapelsden, J. Hosking, and J. Grundy. Design pattern modelling and instantiation using DPML. In *Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications*, pages 3–11. Australian Computer Society, Inc., 2002.

[104] A. Martens, H. Koziolek, S. Becker, and R. Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. page 105. ACM Press, 2010.

[105] F. Massacci, J. Mylopoulos, and N. Zannone. Security Requirements Engineering: The SI* Modeling Language and the Secure Tropos Methodology. In *Advances in Intelligent Information Systems*, number 265 in Studies in Computational Intelligence, pages 147–174. Springer Berlin Heidelberg, 2010.

[106] J. McAffer, J.-M. Lemieux, and C. Aniszczyk. *Eclipse Rich Client Platform*. Addison-Wesley Professional, 2nd edition, 2010.

[107] G. McGraw. The security lifecycle-the 7 touchpoints of secure software-just as you can't test quality into software, you can't bolt security features onto code and expect it to become hack-proof security. In *Software Development*, volume 13, pages 42–43, 2005.

[108] G. McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

[109] A. Mehiaoui, E. Wozniak, S. T. Piergiovanni, C. Mraidha, M. D. Natale, H. Zeng, J.-P. Babau, L. Lemarchand, and S. GÃ©rard. A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems. In *SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, LCTES'13, pages 121–132, 2013.

[110] J. D. Meier, A. Mackman, M. Dunner, S. Vasireddy, R. Escamilla, and A. Murukan. Improving web application security: threats and countermeasures. *Microsoft Corporation*, 3, 2003.

[111] D. Mellado, C. Blanco, L. E. Sánchez, and E. Fernández-Medina. A Systematic Review of Security Requirements Engineering. *Comput. Stand. Interfaces*, 32(4):153–165, June 2010.

[112] Microsoft. Microsoft Security Development Lifecycle (SDL) version 5.2, 2012.

[113] J. Miller and J. Mukerji. Mda guide version 1.0.1. Technical report, Object Management Group (OMG), 2003.

[114] T. M. Mir, A. K. V. Revuru, D. J. Manohar, and V. Batta. *Threat analysis and modeling during a software development lifecycle of a software application.* Google Patents, Jan. 2012.

[115] N. Moebius, K. Stenzel, H. Grandy, and W. Reif. SecureMDD: A Model-Driven Development Method for Secure Smart Card Applications. In *International Conference on Availability, Reliability and Security*, ARES'09, pages 841–846, 2009.

[116] A. Motii, B. Hamid, A. Lanusse, and J.-M. Bruel. Guiding the Selection of Security Patterns Based on Security Requirements and Pattern Classification. In *Proceedings of the 20th European Conference on Pattern Languages of Programs*, EuroPLoP '15, pages 10:1–10:17. ACM, 2015.

[117] A. Motii, B. Hamid, A. Lanusse, and J. M. Bruel. Guiding the selection of security patterns for real-time systems. In *21st International Conference on Engineering of Complex Computer Systems*, ICECCS'16, pages 155–164. IEEE, 2016.

[118] A. Motii, B. Hamid, A. Lanusse, and J.-M. Bruel. Towards the integration of security patterns in UML component-based applications. In *Joint Proceedings of the Second International Workshop on Patterns in Model Engineering and the Fifth International Workshop on the Verification of Model Transformation*, volume 1693 of *PAME '16*, pages 2–6. CEUR-WS.org, 2016.

[119] A. Motii, A. Lanusse, B. Hamid, and J.-M. Bruel. Model-Based Real-Time Evaluation of Security Patterns: A SCADA System Case Study. In *Computer Safety, Reliability, and Security - SAFECOMP 2016 Workshops, ASSURE, DECSoS, SASSUR, and TIPS, Trondheim, Norway, September 20, 2016, Proceedings*, volume 9923 of *Lecture Notes in Computer Science*, pages 375–389. Springer, 2016.

[120] D. Mouheb, C. Talhi, M. Nouh, V. Lima, M. Debbabi, L. Wang, and M. Pourzandi. Aspect-Oriented Modeling for Representing and Integrating Security Concerns in UML. In *Software Engineering Research, Management and Applications*, number 296 in Studies in Computational Intelligence, pages 197–213. Springer Berlin Heidelberg, 2010.

[121] C. Mraidha, S. Tucci-Piergiovanni, and S. Gérard. Optimum: a MARTE-based methodology for schedulability analysis at early design stages. *SIGSOFT Softw. Eng. Notes*, 36(1):1–8, 2011.

[122] P. H. Nguyen, K. Yskout, T. Heyman, J. Klein, R. Scandariato, and Y. L. Traon. SoSPa: A system of Security design Patterns for systematically engineering secure systems. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 246–255, Sept. 2015.

[123] J. Noble. Classifying Relationships Between Object-Oriented Design Patterns. In *Proceedings of the Australian Software Engineering Conference*, ASWEC '98, pages 98–. IEEE Computer Society, 1998.

[124] OBEO. Acceleo. `http://www.eclipse.org/acceleo/`, 2014. [Accessed: January-2015].

[125] OMG. OMG Systems Modeling Language (OMG SysML). `http://www.omg.org/spec/SysML/1.1/`, 2008. [Accessed: May-2014].

[126] OMG. OCL 2.2 Specification. `http://www.omg.org/spec/OCL/2.2`, 2010.

[127] OMG. MOF QVT 1.1 Specification. `http://www.omg.org/spec/QVT/1.1`, 2011. [Accessed: June-2014].

[128] OMG. UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE), Version 1.1. `http://www.omg.org/spec/MARTE/1.1/`, 2011. [Accessed: January-2013].

[129] OMG. MetaObject Facility 2.4.2, Specification. `http://www.omg.org/spec/MOF/2.4.2/`, 2014. [Accessed: July-2014].

[130] OMG. UML 2.5. `http://www.omg.org/spec/UML/2.5/`, 2015. [Accessed: April-2016].

[131] OMG. Unified Component Model For Distributed, Real-Time And Embedded Systems Version 1.0 - Beta1. `http://www.omg.org/spec/UCM/1.0/Beta1/`, 2016. [Accessed: January-2017].

[132] A. L. Opdahl and G. Sindre. Experimental comparison of attack trees and misuse cases for security threat identification. *Inf. Softw. Technol.*, 51(5):916–932, 2009.

[133] OWASP. OWASP CLASP V.A.2. Technical report, Nov. 2007.

[134] D. M. Powers. Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation. *Journal of Machine Learning Technologies*, pages 37–63, 2011.

[135] A. Radermacher, B. Hamid, M. Fredj, and J.-L. Profizi. In *Proceedings of the 18th European Conference on Pattern Languages of Program*, EuroPLoP '13, pages 8:1–8:16. ACM, 2013.

[136] J. F. Ruíz, M. Arjona, A. Maña, and N. Carstens. Secure engineering and modelling of a metering devices system. In *2013 International Conference on Availability, Reliability and Security*, SecSE'13, pages 418–427. IEEE, 2013.

[137] SAE. Architecture Analysis & Design Language (AADL). http://www.sae.org/technical/standards/AS5506A, 2009. [Accessed: July-2014].

[138] C. Sant'Anna, E. Figueiredo, A. Garcia, and C. J. Lucena. On the modularity of software architectures: A concern-driven measurement framework. In *European Conference on Software Architecture*, pages 207–224. Springer Berlin Heidelberg, 2007.

[139] D. Schmidt. Model-driven engineering. *in IEEE computer*, 39(2):41–47, 2006.

[140] B. Schneier. *Secrets & Lies: Digital Security in a Networked World*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2000.

[141] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns: Integrating security and systems engineering*. John Wiley & Sons, 2013.

[142] SEMCO. System and software Engineering for embedded systems applications with Multi-COncerns support. `http://www.semcomdt.org`, 2010. [Accessed: May-2014].

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

[144] A. Shostack. Experiences threat modeling at microsoft. In *Proceedings of the Workshop on Modeling Security*, volume 413, pages 5:1–5:12. CEUR-WS.org, 2008.

[145] G. Sindre and A. L. Opdahl. Eliciting security requirements by misuse cases. In *37th International Conference on Technology of Object-Oriented Languages and Systems, 2000. TOOLS-Pacific 2000. Proceedings*, pages 120–131, 2000.

[146] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: A Flexible Real Time Scheduling Framework. In *Proceedings of the 2004 Annual ACM SIGAda International Conference on Ada: The Engineering of Correct and Reliable Software for Real-time & Distributed Systems Using Ada and Related Technologies*, SIGAda '04, pages 1–8. ACM, 2004.

[147] N. I. O. Standards and Technology. *NIST Special Publication 800-53 Information Security*. CreateSpace, Paramount, CA, 2011.

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

[149] G. Stoneburner, A. Goguen, and A. Feringa. Risk management guide for information technology systems. *Nist special publication*, 800(30):800–30, 2002.

[150] C. Szyperski. *Component Software: Beyond Object-oriented Programming*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998.

[151] Technical Information Bulletin 04-1. Supervisory Control and Data Acquisition (SCADA) System. `https://scadahacker.com/library/Documents/ICS_Basics/SCADA%20Basics%20-%20NCS%20TIB%2004-1.pdf`, 2004. [Accessed: April-2016].

[152] TERESA Consortium. TERESA Project (Trusted Computing Engineering for Resource Constrained Embedded Systems Applications). `http://cordis.europa.eu/project/rcn/93271_en.html`. [Accessed: September-2014].

[153] K. Tindell. *Adding Time-offsets to Schedulability Analysis*. University of York, Department of Computer Science, 1994.

[154] M. Torngren, D. Chen, and I. Crnkovic. Component-based vs. model-based development: a comparison in the context of vehicular embedded systems. In *31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 432–440, 2005.

[155] A. V. Uzunov, E. B. Fernandez, and K. Falkner. Securing distributed systems using patterns: A survey. *Computers & Security*, 31(5):681–703, 2012.

[156] R. Vanciu and M. Abi-Antoun. Finding architectural flaws using constraints. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 334–344, 2013.

[157] J. Viega. Building Security Requirements with CLASP. In *Proceedings of the 2005 Workshop on Software Engineering for Secure SystemsBuilding Trustworthy Applications*, SESS '05, pages 1–7. ACM, 2005.

[158] L. Vogel. Eclipse RCP. `http://www.vogella.de/articles/EclipseRCP/`, 2015. [Accessed: August-2016].

[159] M. Walker, M.-O. Reiser, S. T. Piergiovanni, Y. Papadopoulos, H. Lönn, C. Mraidha, D. Parker, D.-J. Chen, and D. Servat. Automatic optimisation of system architectures using EAST-ADL. *Journal of Systems and Software*, 86(10):2467–2487, 2013.

[160] M. Weiss and H. Mouratidis. Selecting Security Patterns that Fulfill Security Requirements. In *16th IEEE International Requirements Engineering, 2008. RE '08*, pages 169–172, Sept. 2008.

[161] A. G. Wermann, M. C. Bortolozzo, E. G. d. Silva, A. Schaeffer-Filho, L. P. Gaspary, and M. Barcellos. ASTORIA: A framework for attack simulation and evaluation in smart grids. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 273–280, Apr. 2016.

[162] M. Woodside, D. C. Petriu, D. B. Petriu, J. Xu, T. Israr, G. Georg, R. France, J. M. Bieman, S. H. Houmb, and J. Jürjens. Performance analysis of security aspects by weaving scenarios extracted from UML models. *Journal of Systems and Software*, 82(1):56–74, 2009.

[163] J. Yoder and J. Barcalow. Architectural patterns for enabling application security. In *Proceedings of the 20th European Conference on Pattern Languages of Programs*, volume 51. ACM, 1998.

[164] A. Ziani. *Modeling of Secure Dependable (S&D) applications based on patterns for Resource-Constrained Embedded Systems (RCES).* phdthesis, Université Toulouse le Mirail - Toulouse II, 2013. [Accessed: June-2014].