Linköping Studies in Science and Technology

Dissertations. No. 1715

Security in Embedded Systems

A Model-Based Approach with Risk Metrics

by

Maria Vasilevskaya

Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2015

ISBN 978-91-7685-917-9
ISSN 0345–7524

Cover art together with Dmitry Shipilov and Ivan Ukhov

Printed by LiU Tryck 2015

URL: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-122149

To my family

Abstract

The increasing prevalence of embedded devices and a boost in sophisticated attacks against them make embedded system security an intricate and pressing issue. New approaches to support the development of security-enhanced systems need to be explored. We realise that efficient transfer of knowledge from security experts to embedded system engineers is vitally important, but hardly achievable in current practice. This thesis proposes a Security-Enhanced Embedded system Design (SEED) approach, which is a set of concepts, methods, and processes that together aim at addressing this challenge of bridging the gap between the two areas of expertise.

We introduce the concept of a Domain-Specific Security Model (DSSM) as a suitable abstraction to capture the knowledge of security experts in a way that this knowledge can be later reused by embedded system engineers. Each DSSM characterises common security issues of a specific application domain in a form of security properties linked to a range of solutions. Next, we complement a DSSM with the concept of a Performance Evaluation Record (PER) to account for the resource-constrained nature of embedded systems. Each PER characterises the resource overhead created by a security solution, a provided level of security, and other relevant information.

We define a process that assists an embedded system engineer in selecting a suitable set of security solutions. The process couples together (i) the use of the security knowledge accumulated in DSSMs and PERs, (ii) the identification of security issues in a system design, (iii) the analysis of resource constraints of a system and available security solutions, and (iv) model-based quantification of security risks to data assets associated with a design model. The approach is supported by a set of tools that automate certain steps.

We use scenarios from a smart metering domain to demonstrate how the SEED approach can be applied. We show that our artefacts are rich enough to support security experts in description of knowledge about security solutions, and to support embedded system engineers in integration of an appropriate set of security solutions based on that knowledge. We demonstrate the effectiveness of the proposed method for quantification of security risks by applying it to a metering device. This shows its usage for visualising of and reasoning about security risks inherent in a system design.

This work was supported by the Swedish National Graduate School of Computer Science (CUGS), the EU FP7 SecFutur project, and the RICS research centre on Resilient Information and Control Systems.

Populärvetenskaplig sammanfattning

För varje dag som går blir samhället mer beroende av informationsteknologin och betydelsen av inbyggda system i dessa kritiska infrastrukturer ökar. Inbyggda system kan hittas inom en mängd olika domäner. Dessa relativt små system finns i bl. a. hemelektronik, fordon, medicinsk utrustning och industriella styrsystem. I takt med en allt större närvaro av inbyggda system inom olika tillämpningsområden kan vi också se hur sådana system kopplas upp till nätet. Denna utveckling möjliggör skapandet av ytterligare användbara tjänster som ökar tryggheten i vårt samhälle. Tack vare nätanslutning kan exempelvis fordon få assistans vid stora vägkorsningar för att minska risken för olycksfall och en internetuppkopplad pacemaker ger läkarna möjlighet att kontinuerligt kontrollera patienters hälsa.

Den ökande förekomsten av nätverkande inbyggda system ökar dessutom betydelsen av säkerhet för dessa system. Idag är inbyggda system dessvärre öppna för attacker som kan skada andra delar av infrastrukturen. Ett bra exempel på detta är den välkända Stuxnet-attacken år 2010 som påverkade kärntekniska anläggningar i flera länder. Användningen av inbyggda system som är osäkra ökar alltså risken för allvarlig IT-brottslighet som kan leda till betydande risker för samhället.

Den moderna praxisen att lägga till säkerhetsåtgärder sent i utvecklingsprocessen ger inte önskvärt resultat. Detta blir ännu allvarligare när det gäller inbyggda system. Det är ofta svårt eller till och med omöjligt att uppdatera ett inbyggt system när det är redan i bruk. Exempelvis kan det inbyggda systemet ha en kritisk funktion och därför får det inte stoppas för uppdateringen. En annan allvarlig orsak är att inbyggda system är resursbegränsade enheter och därför har de inte någon extra kapacitet att utföra säkerhetsåtgärder när de redan är tillverkade.

I detta sammanhang behövs ett nytt förhållningssätt för att stödja utvecklingen av säkra system. Effektiv överföring av kunskap från säkerhetsspecialisterna till de ingenjörer som bygger de inbyggda systemen, är avgörande men knappast uppnåelig i dagsläget. Denna avhandling föreslår en ansats, benämnd SEED (Security-Enhanced Embedded system Design), som består av en mängd koncept, metoder, och verktyg för att möta utmaningen att bygga en bro mellan de två expertområdena. SEED stödjer säkerhetsspecialisterna genom att tillhandahålla medel för att forma och lagra kunskap om säkerhet, och stödjer ingenjörerna genom metoder att använda och återanvända den lagrade kunskapen vid design av systemen.

Acknowledgement

I would like to express my gratitude to my supervisor Simin Nadjm-Tehrani for the support that she provided to me over these years. Her tireless assistance and guidance have helped me to learn and progress with my research.

I gratefully acknowledge Linda Ariani Gunawan, Peter Herrmann, David Broman, Kristian Sandahl, Ahmed Rezine, and Oleg Sysoev with whom I had many fruitful discussions regarding my research and teaching. Special thanks go to David Broman for his unlimited support and inspiration as my mentor. Thanks to my secondary advisors Nahid Shahmehri and Kristian Sandahl for their feedback given at our meetings and when proofreading this thesis.

I wholeheartedly thank all current and former members of RTSLab for their friendship, support, and all the valuable comments during numerous RTSLab meetings. My appreciation extends also to members of other divisions of IDA with whom I happened to interact regarding my research and teaching.

I would like to thank all administrative personnel who make our working environment very pleasant. Special thanks go to Anne Moe, Eva Pelayo Danils, and Åsa Kärrman. It would have been much more difficult to work effectively without their professional support and patience.

I cannot help but express my gratitude to members of the Karate club from Linköpings budoklubb. We spent an enormous amount of hours training together and making our club a great place to practise Kyokushin.

Last but not least, I am thankful to my family for their support throughout these years. The encouragement and valuable assistance provided by Anatoly and Dmitry helped a lot during these years. I am sincerely thankful to my parents Elena and Viktor as well as to others for their care.

Undoubtedly, there were many other people who contributed to my work with their support, advice or rewarding moments spent together. Unfortunately, it is not feasible to name everyone. Therefore, I anonymously thank all of you who did not find their name here but has contributed with their effort and time at any point of this journey.

Maria Vasilevskaya
Linköping, Sweden
November, 2015

Contents

List of Figures

List of Tables

List of Acronyms

CL	Confidentiality Loss.
IL	Integrity Loss.
AES	Advanced Encryption Standard.
DES	Data Encryption Standard.
DSSM	Domain-Specific Security Model.
EMF	Eclipse Modelling Framework.
GCM	Generic Component Model.
GEM	General Evaluation Model.
GQAM	Generic Quantitative Analysis Modeling.
GRM	Generic Resource Modeling.
HLAM	High-Level Application Modeling.
HRM	Hardware Resource Modeling.
MARTE	Modelling and Analysis of Real-Time Embedded systems.
NFP	Non-Functional Property.
OWL	Web Ontology Language.
PAM	Performance Analysis Model.
PER	Performance Evaluation Record.
RBB	Reusable Building Block.
SAM	Schedulability Analysis Model.
SBB	Security Building Block.
SecFutur	Design of Secure and Energy-efficient Embedded Systems for Future Internet Applications.
SEED	Security-Enhanced Embedded system Design.
SMC	Semi-Markov Chain.

SPACE	SPecification by Activities, Collaborations and External state machines.
SRM	Software Resource Modeling.
TSM	Trusted Sensor Modules.
TSMC	Trusted Sensor Module Collector.
TSN	Trusted Sensor Network.
UML	Unified Modeling Language.

1 Introduction

The ubiquitous presence of networked embedded devices comes as no surprise. Large computing infrastructures that bring automation in our daily lives exist due to support of such devices interconnected through networks. Being a part of such infrastructures, embedded devices carry and process sensitive information. Thus, both their exposure to open networks and their critical role in storing, processing, and transmission of information makes embedded devices a target of sophisticated attacks. The interest of attackers is stimulated by the fact that modern embedded systems are often easily accessible (e.g. deployed in a public and untrusted environment) and the consequences of compromising such devices can be very large. If an attacker hacks a device of one type, the attack can be quickly replicated to all other devices of the same type possibly in thousands or millions. If attackers take control over one of the devices of a large network, they can gain access to other devices of the network. For example, a computer scientist at Colombia University, Ang Cui, has developed a technique that allows taking complete control of a Cisco IP phone, that in turn allows affecting other parts of a connected system (other phones in a network, computers, printers, etc.) [4]. These facts impose high requirements on security standards for embedded systems that are often neglected. To emphasise the point, McClure estimates [5] that there are already 10 billion embedded devices in operation that were designed without much thought about security.

Thus, it goes without saying that security issues should be considered during embedded system development since insufficient security can create a substantial risk for society and significant loss of profits for embedded system producers, owners, and end users.

1.1 Motivation

Although security is an essential aspect of networked embedded systems, it is still approached as an add-on late in the development process. This can hardly be effective due to the complexity of embedded systems, their resource-constrained nature, and non-functional requirements. For instance, Ravi et al. [6] discuss the main consequences of incorporating security solutions into embedded systems at the late development phases. Resources planned during the initial development phase do not account for security functions. These insufficient resource allocations dramatically limit the number of security solutions available for a system engineer or even put this number to zero. The authors identify a set of bottlenecks that system designers consequently have to deal with. These include, but not limited to, the energy consumption overheads (the battery gap) and the computational demands (the processing gap). Ravi et al. argue for a shift to an appropriate design methodology to address these challenges.

There is a number of factors that make attacks on embedded systems successful. An unthoughtful system design is one of the sources of potential breaches. Vulnerabilities introduced during the implementation phase are another one. Human factors such as an intentional and unintentional misuse of system components are major problems during the usage phase. While all factors are significant, this thesis focuses on resolving security issues at the design phase of the system development. The underlying reason for such a focus is that the consequences of an unthoughtful design influence all later phases of the system life cycle. At the same time, integration of security mechanisms already at the design phase allows early exploration of performance, power consumption, cost and other trade-offs. Practitioners recognise that including security into embedded devices should be a critical design task, and that building security at the early phase into these systems will provide protection that reduces the need for additional security appliances [7]. This motivates adopting the principles of model-based engineering [8] as a vehicle to bring security consideration to a design phase.

A mere focus on the design phase is not enough to efficiently tackle security issues. The challenge is amplified by the diversity and complexity of both security solutions and embedded systems as such. Embedded systems design requires in-depth understanding of an application domain, usage scenarios, and deployment environment. Security threats, in turn, vary from application to application and are more or less prevalent in an application domain. Due to these concerns, Kocher et al. [9] stress the need of system engineers (who are not necessarily experts in security) to understand both required level of security assurance and the overhead caused by integrating security solutions into a system design; while practitioners confirm that engineers are typically not experts in both security and application domains [10].

Security mechanisms should be developed and thoroughly studied by

security experts, whereas the resulting knowledge should be available for embedded system engineers. Generic security solutions are not suitable for the limited resources of embedded systems. A security solution for embedded systems should be specific for a particular application domain in order to provide the required efficiency at the cost of acceptable performance. Last but not least, there are far fewer security experts than embedded system engineers. These factors together motivate two principles that we exploit in this thesis towards improving practices of security-enhanced embedded system development, namely the separation of roles (i.e. an embedded system engineer and security expert) and domain specialisation (i.e. application-driven security) principles.

Very often security is in conflict with other complex requirements on the functionality and performance of an embedded system. Dealing with security is also hard because there are a lot of actors and phenomena affecting security of a system and that cannot be fully controlled and accounted for. One example is incomplete knowledge about adversary behaviour [11]. Therefore, providing perfect security is far beyond our reach and almost an unattainable goal [12]. This dictates that security is not a binary property. Instead, security should be approached as a measurable quantitative property. A quantitative notion of security can indicate a degree of protection, and thus, decision makers can be equipped with tools for reasoning about the trade-off between security and other constraints (functional requirements, system resources, economic factors, etc.).

Measured security can help answering different questions and in this thesis we are mostly concerned with questions like: Given particular design alternatives which of them provides higher level of security? Given several security countermeasures each with associated integration costs which one is the most beneficial to implement? Answering the former question can support system engineers to systematically improve a system design and to reduce associated security risks. The latter question has a special importance for parties affected by a designed embedded system (stakeholders). In particular, answering the second question enables conducting cost-benefit analysis that provides input information for distributing additional costs associated with integration of security features. A part of the work in this thesis is concerned with exploring ways to quantify security of embedded systems at the design phase.

1.2 Problem Formulation

The objective of this thesis is to provide concepts and tools for addressing security issues of embedded systems already at the design phase. We aim to reach this by defining an approach which targets two categories of professionals. With the help of the developed approach, security experts should have an opportunity to describe developed security solutions in a reusable manner. This, in turn, should enable embedded system engineers to select

a suitable set of security solutions based on the analysis of both system's security needs and system resource constraints. The approach explores the following principles:

- *Model-orientation*: which allows dealing with security concerns already at the early development phase.
- *Domain specialisation*: which increases the quality and efficiency of eventual solutions by narrowing down the focus to a specific domain.
- *Separation of responsibilities and concerns*: which supports reuse of security solutions by promoting separation of the security expert and embedded system engineer roles.

Adopting the basic principles stated above, we contribute to tackling the challenge of providing support for a security expert and an embedded system engineer by answering the following research questions:

1. *What abstractions are suitable for a security expert to assist in creating a useful description of a security mechanism?* To enable reuse of security knowledge, first it is necessary to define a suitable set of concepts to capture relevant information. Usefulness of this description means that it should provide sufficient information for system engineers to make informative decisions.
2. *What technologies and processes can be employed to assist a security expert in capturing this knowledge?* Adequate support is an important factor that enables use of the developed concepts. This support should be provided by appropriate modelling structures and a process for security experts that will assist these specialists in capturing security knowledge by using these modelling structures.
3. *What are methods and tools that should equip an embedded system engineer to enable the use of the provided security knowledge?* These methods and tools should assist a system engineer in exploring alternative ways and selecting a set of security solutions to secure a system under development by using the security knowledge provided by security experts. Complementary to evolving best practices, and development of new countermeasures, our work aims at analysing possible alternatives with respect to a certain system design.

1.3 Contributions

The main contribution of this work is the definition of a **S**ecurity-**E**nhanced **E**mbedded system **D**esign (SEED) approach with the associated tools and methods. The contributions of this work are described more specifically below.

1. *Two concepts that represent the domain-specific security knowledge.*

   SEED rests on two concepts that encapsulate a rich set of information about security solutions. These are **D**omain-**S**pecific **S**ecurity **M**odel (DSSM) and **P**erformance **E**valuation **R**ecord (PER). DSSM allows capturing the functional specifications of a security solution to the extent it is needed for integration of these solutions into a system model. Besides, each solution is annotated with the information about what security issues this solution solves and its relation to other security mechanisms. PER serves to associate information about performance characteristics and indicators with a security mechanism. Together DSSM and PER contain the rich information about capabilities and demands of a security solution that are important to consider when a system engineer makes decisions on needed protection.

2. *Ontology- and model-based framework that implements the introduced concepts.*

   Having introduced the two concepts (DSSM and PER), we face the need to provide a means to support their usage. We achieve this by representing DSSM and PER as UML models. Thus, for capturing security knowledge a security expert simply needs to instantiate corresponding UML models. As we mentioned, these concepts contain rich information of a diverse sort about security countermeasures. To operate with this information, we utilise the ontology technology as a viable technology for storing and managing the captured security knowledge.

3. *A process elaborated for a security expert to capture security knowledge specific for a certain application domain.*

   As part of this contribution, we elaborate a process that guides a security expert on how to use the developed concepts. Each step of this process in SEED is supported by the developed MagicDraw [13] plug-in.

4. *A process elaborated for a system engineer to select security countermeasures.*

   Similarly, we define a process to be followed by a system engineer when incorporating security countermeasures into a system design. The process is elaborated using three methods: asset elicitation technique, model-based compatibility analysis, and a method for quantification of security risks to data assets.

   - *Asset elicitation technique*

     This method allows analysing a system design for identification of security needs of a system.

- *Model-based compatibility analysis*

  This method contributes in matching resource constraints of an embedded system under development with demands of different security solutions.

- *Quantification of risks to data assets*

  This method provides two probabilistic risk-based metrics, i.e. *confidentiality loss* and *integrity loss*, to quantify security of a system with respect to data assets in the context of a given design model. This includes development of a formal method for derivation of these metrics using three types of models as inputs: functional model of a system, its execution platform model, and attack models. This method is an integral part of SEED, however, its application goes beyond SEED.

These methods building the system engineer process of SEED are also supported by a set of tools integrated into the MagicDraw environment. Besides, the method for quantification of risks to data assets is supported by a Python-based tool.

5. *Application of the developed concepts, methods, and processes on scenarios from the metering infrastructure domain.*

   Throughout the thesis we use scenarios from the smart metering infrastructure application to instantiate and demonstrate the introduced concepts and methods. Consequently, we have demonstrated that SEED is able to support an embedded system engineer to integrate a suitable set of security solutions exploring the captured security knowledge. Moreover, we have implemented a smart meter prototype that we use as an illustrative example for analysing risks to data assets.

1.4 Research Method

A fundamental question underlying computer science is "What can be (efficiently) automated?" [14]. The research conducted in this thesis can be classified as technology research [15] that is concerned with creating new or improving existing artefacts. Solheim and Stølen [15] summarise the technology research as an iterative process built of three steps: problem analysis, innovation, and evaluation. The problem analysis step is concerned with identifying the potential need for an improved or new artefact. Consequently, the innovation step deals with invention and creation (improvement) of an artefact satisfying the needs. At the evaluation step, a researcher tries to find out whether the produced artefact satisfies the formulated needs.

McGrath [16] distinguishes eight evaluation strategies. These are field study, field experiment, experimental simulation, laboratory experiment,

qualitative interview, survey, non-empirical evidence, and computer simulations. Among other discussions, each strategy is analysed with respect to three desired properties. These are: precision (that the conclusions are precise), generalisability (that the results are valid across a population), and realism (that the evaluation is performed in an environment similar to reality). Ideally one would prefer to apply such an evaluation strategy that provides the highest precision, the highest generalisability, and the highest realism. However, the author [16] concludes that none of the strategies can score the highest with respect to these three properties in practice. For example, a laboratory experiment has high precision, but may lack realism and generalisability. A survey scores high on generalisability, but less on precision and realism.

We adopt the research method outlined above iterating over problem analysis, innovation, and evaluation. The problem is formulated as a result of analysing the scientific literature and investigating three industrial case studies together with industrial and academic partners within the EU SecFutur project [17]. Our evaluation is based on three studies:

- Study 1: Application of the SEED constituents on scenarios from the metering infrastructure domain provided by industrial partners from the EU SecFutur project.
- Study 2: Application of the SEED constituents on a smart meter prototype developed based on the design from the EU SecFutur project.
- Study 3: Analytical evaluations of the suitability of SEED for its intended use.

Study 1 and Study 2 can be considered as variants of the laboratory experiment approach described by McGrath [16]. Study 1 is reported in Chapters 4 – 5, and Study 2 is applied in Chapter 6. Study 3 used in Chapter 5 is a variant of the non-empirical evidences approach that scores the most on generalisability. Moreover, each piece of work has been evaluated with respect to the scientific literature (see Chapter 7). These studies are complemented by evaluations through discussions and workshops with senior colleagues, academic and industrial partners within the European SecFuture project [17], and by obtaining reviews from the research community when publishing the results of this work.

1.5 List of Publications

The work presented in this thesis is based on the following publications:

- Maria Vasilevskaya, Linda Ariani Gunawan, Simin Nadjm-Tehrani, and Peter Herrmann, Security Asset Elicitation for Collaborative Models, in Model-Driven Security Workshop (MDSec) in conjunction with MoDELS, ACM, Innsbruck, Austria, pp 7:1–7:6, October 2012.

- Maria Vasilevskaya, Linda Ariani Gunawan, Simin Nadjm-Tehrani, and Peter Herrmann, Integrating Security Mechanisms into Embedded Systems by Domain-specific Modelling, Journal of Security and Communication Networks 7(12): 2815–2832 (2014), Wiley, 2014.

- Maria Vasilevskaya and Simin Nadjm-Tehrani, Model-based Security Risk Analysis for Networked Embedded Systems, in the International Conference on Critical Information Infrastructures Security (CRITIS), Springer, Limassol, Cyprus, October 2014.

- Maria Vasilevskaya and Simin Nadjm-Tehrani, Support for Cross-domain Composition of Embedded Systems Using MARTE Models, ACM SIGBED Review – Special Issue 12(1): 37-45, 2015.

  This article is an adaptation of an earlier version presented in the workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS) at RTSS, Vancouver, Canada, December 2013.

- Maria Vasilevskaya and Simin Nadjm-Tehrani, Quantifying Risks to Data Assets Using Formal Metrics in Embedded System Design, in the International Conference on Computer Safety, Reliability and Security (SAFECOMP), Springer, Delft, the Netherlands, pp 347 – 361, September 2015.

Some content of this thesis has been published as parts of deliverables 3.1, 4.1, 4.2, and 4.3 of the EU FP7 SecFutur project [17].

The following papers co-authored in parallel with the presented work are not included in this thesis:

- Simin Nadjm-Tehrani and Maria Vasilevskaya, Towards a Security Domain Model for Embedded Systems, in the IEEE International Symposium on High Assurance Systems Engineering (HASE), poster session, Boca Raton, FL, USA, pp 180 – 181, November 2011.

- Maria Vasilevskaya, David Broman, and Kristian Sandahl, An Assessment Model for Large Project Courses, ACM Technical Symposium on Computer Science Education (SIGCSE), Atlanta, GA, USA, pp 253 – 258, March 2014.

- Maria Vasilevskaya, David Broman, and Kristian Sandahl, Assessing Large Project Courses: Model, Activities, and Lessons Learned, ACM Transactions on Computing Education, 2015.

- Klervie Toczé, Maria Vasilevskaya, Simin Nadjm-Tehrani, Patrik Sandahl, Maintainability of Functional Reactive Programs in a Telecom Server Software. Submitted.

1.6 Outline

This thesis is composed of eight chapters where Chapters 3 – 6 contain the core of this work. We provide the necessary background to our work in Chapter 2. Chapter 3 explains the idea and structure of SEED in general terms that are detailed in the following chapters. In particular, Chapter 4 defines the process for capturing of the domain-specific security knowledge. Chapter 5 explains methods and tools that support the use of the captured knowledge for integration of security countermeasures into an embedded system design. We focus on presenting two metrics for quantifying security risks associated with a system design in Chapter 6. A reader may turn to Chapter 7 to see the relation of our work to other works in the areas of system engineering, security engineering, and embedded systems. Finally, Chapter 8 concludes this thesis and gives some pointers for future work. Figure 1.1 summarises how the problem formulated in Section 1.2 is covered by the contributions and the structure of this thesis.

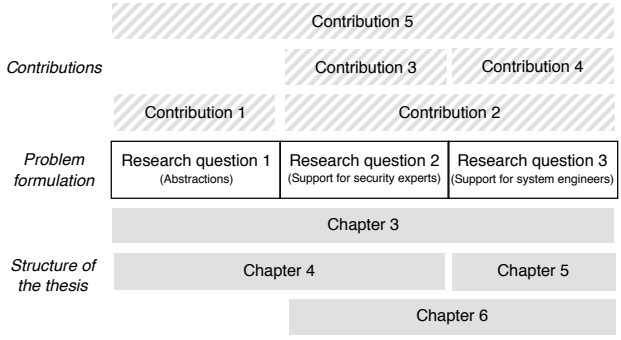


Figure 1.1: Structure of the thesis

2 Background

This chapter provides the necessary background needed in the context of this work. Section 2.1 gives an overview of the basic process models for embedded system engineering. Then, the basic concepts, tools, and methods of model-based engineering are introduced in Section 2.2 followed by a brief introduction to ontology technologies given in Section 2.3. Section 2.4 introduces semi-Markov chains. Finally, we conclude this chapter with Section 2.5 by presenting a case from the smart metering domain used for application of the concepts, methods, and processes developed in SEED. We also use this case as a running example throughout this thesis to illustrate the introduced artefacts.

2.1 Embedded Systems Engineering

Development of embedded systems is a complex task. Therefore, a set of process models exist that support engineers in tackling this complexity. In a broad sense, a process defines a set of activities, their input/output artefacts, roles with responsibilities, time frames, and costs. In our work, we are mainly concerned about activities and input/output artefacts.

At the level of main activities, life cycle models (i.e. process models) for development of embedded systems are very similar to life cycle models proposed for general software engineering. In particular, there are five basic steps that span across the whole life cycle of a system: requirements definition, system specification, functional design, architectural design, and prototyping/implementation [18]. The first step intends to capture the customer's true needs in terms of what a system shall do. The following step,

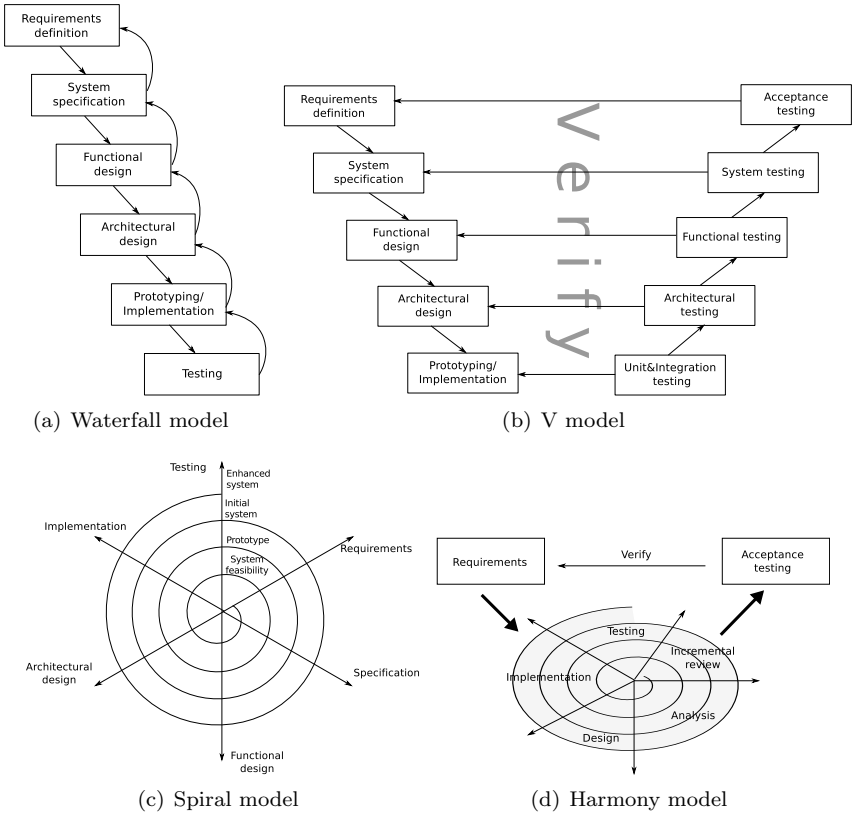
i.e. system specification, refines the customer description in a more concise and precise form. The next two steps go deeper and turn specifications into a set of functional blocks that are later mapped into architectural elements. These elements are combinations of hardware and software resources. Finally, a system is implemented that results in a prototype. The extended life cycle models instrument these basic five steps with extra activities, such as testing, validation, verification, and maintenance. In the following, we outline three widely known life cycle models, namely waterfall, spiral, and V models.

- The *waterfall* [19] model depicted in Figure 2.1(a) represents a development cycle as a sequence of the steps above. According to this model, an engineer should proceed to the next step when the current phase is completed. Additionally, there is a feedback loop (depicted by the backward arrows) to the previous phase that ensures conformance of artefacts created on the current phase to the artefacts produced on the previous step. The presence of this feedback loop differs the waterfall model from a simple sequential process.

- The *V* model [20] depicted in Figure 2.1(b) is similar to the waterfall model, but it emphasises the verification and validation (V&V) activities. A system development follows the top-down approach (the left-hand side), while the verification and validation activities go from the bottom to the up (the right-hand side). Thus, the implemented system is verified against each produced artefact, namely implementation, architectural design, functional design, specification, and requirements. Unit and integration testing verifies a system against the artefacts created at the prototyping/implementation phase, e.g. program design. Thus, in this process model each activity in the development leg (the left side in Figure 2.1(b)) has a counterpart on the same abstraction level in the V&V leg (the right side in Figure 2.1(b)).

- The *spiral* model [21] depicted in Figure 2.1(c) promotes an iterative style for development of a system. Thus, the main difference of the spiral model and the above mentioned models is that it emphasises iterative emergence of several versions of the same system. First, a very restrictive version of a system is developed to understand if the requirements are correctly and adequately formulated. Then, the system evolves into more complex and complete versions, e.g. prototype, initial design, and enhanced design. The corresponding artefacts, e.g. requirement specifications, functional and architectural designs, implementation, also evolve. The radius of the spiral can reflect the amount of time spent on each cycle.

Different variations, modifications, and combinations of the presented process models exist. For example, Douglass [22] proposes a so called *har-*



(a) Waterfall model (b) V model

(c) Spiral model (d) Harmony model

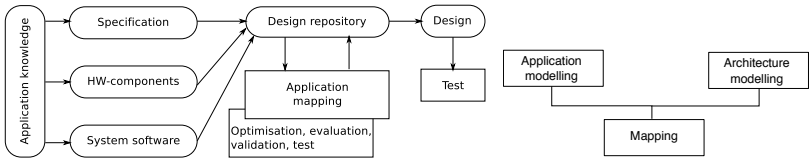
Figure 2.1: Life cycle process models

mony development process (see Figure 2.1(d)) where the V and spiral models are combined.

Hardware/software partitioning is a important task of the embedded system development that differentiates these systems from software-centric systems. System partitioning can rely on the Y-chart approach [23, 24] depicted in Figure 2.2(b). This approach is built of three main elements: application modelling (the description of system functions), architecture modelling (the description of potential execution platforms), and mapping of the application on modelled architectures (allocation).

There is also a process [25] used in embedded system development that differentiates two distinct levels. They are *system* and *lower* levels. In this approach, verification, validation, testing, estimation, and analysis steps are tightly woven into a process. The *system level* concerns defining a system model and selecting a suitable architecture. These artefacts are further evolved into different parts of code (RTOS and application code) and ele-

ments of hardware at the *lower level*.



(a) Simplified design flow [26] (b) Y-chart

Figure 2.2: Two life cycle models for embedded system development

The last model for the life cycle development that we visualise in this section is the *simplified design flow* presented by Marwedel [26]. Figure 2.2(a) depicts this process. This model does not radically differ from the models presented above. The design starts from some application knowledge that is transformed into specification, and hardware/software components. However, Marwedel explicitly brings the *design repository* into the process. According to Marwedel, this repository serves to keep track of design models evolution. However, we envisage a wider use of this component, namely as a point to extend and refine the initial design. The evolution of this idea is further exploited and evolved in Chapters 3 – 6.

2.2 Modelware Zoo

In this section, we briefly introduce the reader to the area of model-based engineering. First, we cover main concepts of the modelling theory. Thereafter, we describe two modelling languages used in our work, namely UML and MARTE, and an employed system modelling approach called SPACE together with its modelling language. We conclude this section outlining tools that support principles of model-based engineering.

2.2.1 Main Concepts

We begin with introducing terms of models, meta-models, transformation, and basics of the language engineering, followed by brief discussions on topics such as domain-specific compared to general modelling, and model-based compared to model-driven engineering.

Models and Meta-models

In general, models allow to raise the abstraction level to deal with growing complexity of artefacts (e.g. embedded system design) [27]. Abstraction improves understanding of complex artefacts and allows their efficient analysis through hiding some irrelevant information. In other words, a model represents a real system highlighting its properties of interest.

Any model conforms to some meta-model that defines its properties. Thus, a *meta-model* defines a modelling language used to create a model of a certain type for a system. Depending on the type of properties that a model should describe an employed meta-model will change. Consequently, a meta-model conforms to some language used to define properties of this meta-model, i.e. a *meta-meta-model*. In theory, an infinite hierarchy of model-meta-model relations can be specified. However, in practice, meta-meta-model is abstract and general enough to define itself wrapping the *layered organisation* of modelware (see the left side of Figure 2.3). Such organisation is sometimes referred to as the *four-layered architecture (M0-M3)* [28] or *3+1 organisation* [29].

The right side of Figure 2.3 depicts a classical example that demonstrates an instantiation of the layered organisation introduced above. A real-world object (car) is shown at level M0. A model of the car is shown at level M1. This model describes a car as a Car class with one "colour" attribute. The meta-model located at M2 explains how to understand this model, namely what elements are classes and what elements are attributes. Finally, level M3 defines concepts used at level M2. Thus, both Attribute and Class are represented as classes at M3.

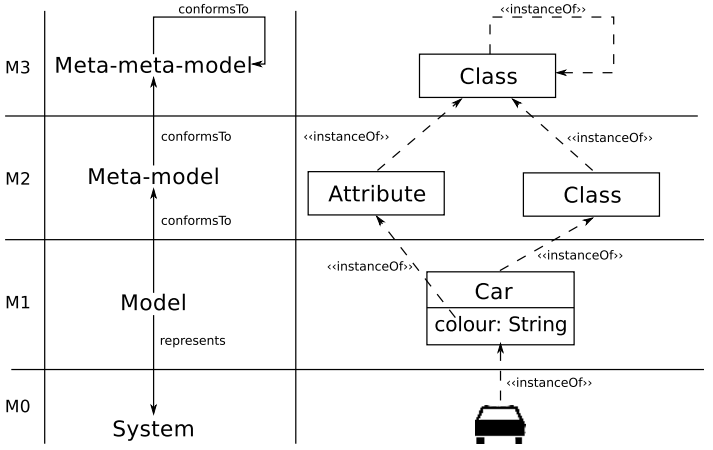


Figure 2.3: Models, meta-models, and meta-meta-models – the layered organisation

The Object Management Group (OMG) [30] implements the M3 level as the Meta-Object Facilities (MOF) standard [31]. MOF is used to define the Unified Modelling Language (UML) [28] located at the M2 level.

Transformation

Model transformation is a technique that allows defining a mapping between different models, i.e. *source* and *target* models, that are different

representations of the same system. Figure 2.4 depicts a classical scheme that explains concepts of model transformation and their relations. Any model transformation is applied to source and target models, but the actual transformation is defined at the meta-model level, i.e. a model transformation definition refers to elements of the meta-models of the source and target models. Thus, model transformation receives input and output models that conform to their respective source and target meta-models. At the same time, a model transformation definition is a model by itself that conforms to some meta-model, i.e. to a transformation language [29, 8].

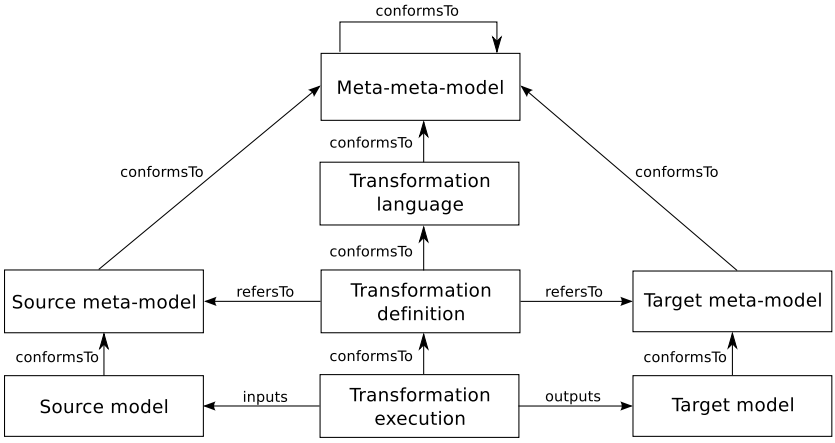


Figure 2.4: Model transformation: concepts and their relations

Transformation languages can be classified as declarative, imperative, and hybrid. Declarative languages require an engineer to specify relations between source and target meta-models, e.g. in terms of functions. In contrast, one needs to specify such details as execution order (sequence of steps) when imperative languages are used. A hybrid type of languages is an intermediate category that mixes constructs and principles from both declarative and imperative languages.

Depending on the nature of target and source meta-models, transformation languages can be classified as model-to-model (M2M) and model-to-text (M2T) transformations [32]. Naturally, the former type of transformations input a model conforming to a certain meta-model (e.g. UML) and produce another model that conforms to a different meta-model (e.g. Entity-Relation Diagram), while the latter type of transformation results in some textual representation (e.g. Java code). Recently, a third type of transformation called text-to-model (T2M) has been introduced. Additionally, one can classify a transformation as *endogenous* or *exogenous*. A transformation is considered to be *endogenous* if source and target models conform to the same meta-model. In contrast, an exogenous transformation is used when source and target models conform to different meta-models.

Model transformation is a powerful concept that is used to automate different tasks of model-based engineering [33]. For example, code generation is a special type of model transformation where the target model is code. Model composition, model refactoring, verification, and reverse engineering are other examples of scenarios where model transformation can be applied.

Abstract Syntax, Concrete Syntax, and Semantics

To enable sophisticated operations with models (e.g. transformation), they must have a well-defined structure. Therefore, techniques for systematic definition of meta-models should be used. The research area that concerns proper definition of complex modelling languages is sometimes referred to as *modelling language engineering* [34].

The main elements that define a language are its *syntax* (i.e. a language's notation) and *semantics* (i.e. a language's meaning). There are two types of syntax that serve for different purposes, namely an *abstract* syntax and a *concrete* syntax [35]. An *abstract syntax* defines all valid models of modelling languages. For example, an abstract syntax defines what are concepts of a modelling language (e.g. classes and their attributes) and what are their valid relations (e.g. associations). Meta-modelling (see Figure 2.3) is a technique for defining an abstract syntax. A *concrete syntax* defines how an abstract syntax appears for an engineer (i.e. for its users). Thus, a concrete syntax deals with representation of a modelling language. A concrete syntax can be represented in textual or visual (e.g. boxes and arrows) notations.

Semantics defines the meaning of a language notation (i.e. syntax). In general, there are two steps to define semantics for a language. First, a *semantic domain* should be defined that provides a meaning for each expression. This meaning must be an element of another well-understood domain, e.g. real numbers. Afterwards, a *semantic mapping* should be created to bound elements of an abstract syntax to a defined semantic domain. GPML

Domain-specific vs. General-purpose Modelling

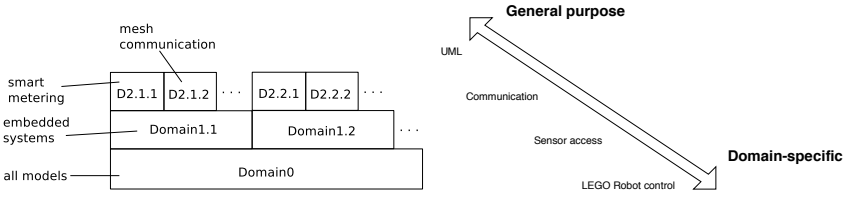
Model-based engineering methods distinguish two big categories of modelling languages, namely *Domain-Specific Modelling Languages* (DSMLs) and *General-Purpose Modelling Languages* (GPMLs). DSMLs are languages that are designed for a certain *domain* [36]. Such languages are usually designed by a group of experts to be used in a specific context or company to facilitate a particular task (e.g. the task of describing things in that domain). In other words, a DSML allows the user to specify a solution using terms of a problem domain that are built in this DSML. Besides, DSMLs are intended to support a better reuse of functionality recurring in a set of modelling tasks. A DSML is optimised for a certain class of problems within a domain. In contrast, a GPML does not target a specific domain, but rather is intended to be applied in any domain.

Since expressiveness of DSMLs is bound to a particular domain, they can be used only for a predefined set of problems; whereas GPMLs are advertised to be suitable for a wide range of modelling tasks. However, DSMLs bring higher productivity and conciseness in modelling since an engineer operates with a limited set of concepts that are familiar and intuitive for a considered domain.

There are a lot of discussions on the topic of "DSML vs. GPML". Both classes of languages are suitable for different purposes and scenarios. Therefore, it is rational to be aware of their advantages and disadvantages through their systematic comparison. Boundaries between domain-specialisation are not obvious: any language is more or less domain-specific [1]. In this section, we outline some characteristics that are typical for a pure DSML and GPML.

A pure DSML can be characterised by the following set of peculiarities: it is designed for a narrow and well-defined domain; it has a relatively small size with a limited set of user-defined abstractions; its development takes months to years; it is designed by a few domain experts; its user community is a narrow and accessible group. In contrast, a pure GPML is designed for a large and complex domain; it has a large language size with a sophisticated set of general abstractions; its development usually spans over years and decades; it is designed by gurus and large communities.

To conclude our discussions about DSMLs and GPMLs, Figure 2.5 (presented by Voelter [1]) illustrates views on relations between domains and languages. Figure 2.5(a) shows the relations between domains as a hierarchical structure where a domain of a pure GPML is the lowest level. An example of a language order based on their domain-specificity is depicted in Figure 2.5(b). We believe that these figures give a good intuition on boundaries between domain-specific and general-purpose modelling languages.



(a) Domain hierarchy (b) Language order

Figure 2.5: Relations between domains and languages, adapted from [1]

Model-based vs. Model-driven Engineering

A set of development paradigms that rely on models as a key artefact have recently emerged. For example, Liebel et al. [37] recently report in their state-

of-practice survey that use of models is widespread in the embedded domain. These paradigms are Model-Based Engineering (MBE), Model-Driven Development (MDD), Model-Driven Engineering (MDE), and Model-Driven Architecture (MDA) that can be distinguished based on the role of models.

To begin with, we briefly explain the difference between "model-driven" and "model-based" prefixes. Intuitively, the latter is a softer version of the former: the former prefix says that models drive the process, while in the latter case models play an important role, but are not key artefacts. In case of "model-driven", it is often expected that a model is used to generate the final implementation. In contrast, for the "model-based" techniques, a model can be used for various kinds of analysis and even test generation, but the actual implementation can be done by developers. Thus, MDE can be considered as a subset of MBE [8]. Similarly, MDD is a subset of MDE, since the letter "D" stands for "Development" that is one type of activity in system engineering. Finally, MDA is a realisation of MDD proposed by Open Management Group (OMG) [30] that is inherently based on OMG standards.

2.2.2 UML

Unified Modelling Language (UML) is a widely accepted general purpose modelling language. Modelling concepts defined by UML are organised in different types of diagrams. The current version of UML (v2.4.1) [28] differentiates 14 types of diagrams. These diagrams are classified in two categories: those that are intended to model *structural* and *behaviour* parts of a system. Each type of diagram uses different modelling concepts that together allow describing diverse aspects of a system.

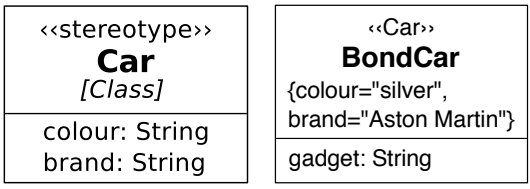
There are seven types of structural diagrams. A class diagram shows system's classes, their attributes, their operations, and the relations among classes. A component diagram shows the components of a system and their relations. A composite structure diagram shows the internal structure of a class and the interaction (collaboration) that this structure enables. A deployment diagram can be used to show the hardware/software parts of a system and artefacts deployed on this execution environment. An object diagram shows instantiation of the system classes. A package diagram shows the logical organisation of a system as a set of packages and their dependencies. Finally, a profile diagram encapsulates custom domain-specific extensions of the standard UML constructs (see below).

The remained seven types of UML diagrams allows describing behaviour of a system. An activity diagram can be used to show a workflow (both control and data) of a system. A state machine diagram shows the system's states and their transitions. A use case diagram is used to give a high-level description of a system in terms of actors, their goals, and dependencies between actors and goals. Communication and sequence diagrams are used to describe the interaction and communication between objects as sequences

of messages using different syntaxes. The last two types of diagrams are interaction overview and timing diagrams that enable creation of an overview of a system and specifying some timing constraints of operations respectively.

This rich set of diagrams have a complex and diverse syntax, but their semantics is rather weak. As a result, UML diagrams are used for modelling tasks in many different domains, but these models are not comparable due to the absence of a commonly agreed semantic domain. Therefore, usually a small subset of UML (i.e. some syntactical constructs such as a subset of UML class diagrams) is used and is further supported by a user-defined semantics bound to a considered domain.

In addition, the UML standard defines extensibility mechanisms that can be used to add domain-specificity to UML. They are *profiles*, *stereotypes* and *tag definitions*. A *profile* is a special type of package that contains stereotypes. A visual representation of a profile is referred to as a profile diagram. A *stereotype* allows adding a set of specific properties (suitable for a particular domain) into existing UML concepts (e.g. class, activity, component). Thus, a stereotype can be considered as a mechanism to refine existing UML concepts with required non-standard semantics. Each stereotype extends (refines) some UML base meta-class (e.g. class, property, named element). Therefore, a stereotype can be used to annotate only those concepts that extend the same meta-class. A stereotype can introduce additional domain-specific properties. These properties are defined through so-called *tag definitions*. Figure 2.6 depicts a small example of a stereotype definition and its usage. Figure 2.6(a) shows a stereotype called *Car* that has two tag definitions, namely *colour* and *brand*. Thereafter, we have applied the stereotype *Car* to specialise the class *BondCar* (see Figure 2.6(b)). Hence, *BondCar* has all the properties declared for the *Car* stereotype. The colour and brand properties can be assigned to some values. For our example in Figure 2.6(b), they are *silver* and *Aston Martin* respectively.



(a) Definition of a stereotype

(b) Usage of a stereotype and tag

Figure 2.6: Example of the stereotype definition and usage

Note, that a stereotype should not be confused with the inheritance relation. Annotation of entities with a certain stereotype does not bring the classical child-parent dependency. In our example, if we remove the

stereotype *Car* from the model, the class *BondCar* will still exist but without the *colour* and *brand* properties that belong to the stereotype *Car*. In contrast, a child can not exist without a parent when the inheritance relation is established.

When modelling a complex system, an engineer may need to use several UML profiles for covering diverse domain-specific characteristics in one model. However, this can create inconsistencies when the used profiles contain concepts that overlap and contradict to each other. Noyrit et al. [38] discusses the issues of using multiple UML profiles and also suggest an approach for resolving identified issues.

One can distinguish two main approaches to defining a UML profile [39]. The first approach starts directly by defining a set of stereotypes that extend the UML meta-model. The second approach introduces a more systematic two-stage process. According to the latter approach, an engineer first needs to create a conceptual model for a domain. A conceptual model describes all (relevant) concepts of a selected domain and their relations. In the second stage, the actual set of stereotypes together with their attributes and constraints is derived from a conceptual model. This process is sometimes referred to as *mapping* [40]. Lagarde et al. [39] suggest to automate this step to avoid errors and to enable relevant verifications ensuring consistency of the resulting profile with its conceptual model. In the context of the modelling language engineering explained in Section 2.2, the mentioned conceptual model can be referred to as an abstract syntax and a profile as a concrete syntax.

2.2.3 MARTE

MARTE [41] is a standardised UML profile designed for Modelling and Analysis of Real-Time Embedded systems. It contains a rich set of concepts to support design and analysis of embedded systems. The structure of this profile is outlined in Figure 2.7. The MARTE foundations package provides a set of concepts required to model non-functional properties (NFP package), time properties (Time package), generic resources of an execution platform (GRM package), and resource allocation (Alloc package). These foundations serve as basics for the MARTE design and analysis models.

The design package contains sub-packages to describe the hardware and software resources, namely Hardware Resource Modeling (HRM) and Software Resource Modeling (SRM). Additionally, the design modelling packages contain concepts to model a component structure and application features. These concepts are encapsulated into the Generic Component Model package (GCM) and High-Level Application Modeling (HLAM) packages. The MARTE analysis package provides facilities to model the context required to perform analysis of real-time and performance characteristics of embedded systems. In particular, the Generic Quantitative Analysis Modeling (GQAM) package defines a set of general terms while its extensions refine

them to support schedulability (SAM) and performance analysis (PAM).

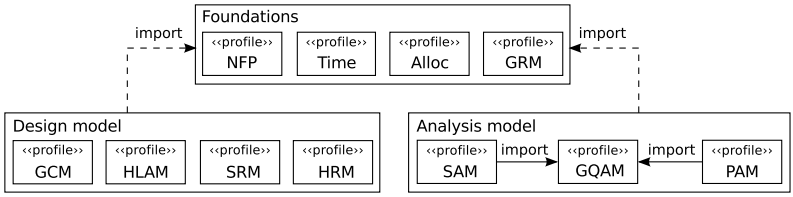


Figure 2.7: Structure of the MARTE profile

Figure 2.8 shows the basic elements of the GQAM package. Its central concept is the *Analysis Context*. This concept aggregates all relevant information needed to describe the constitutents of any type of analysis. In particular, the analysis context concept relates *resource platform* and *workload behaviour* elements. A workload behaviour defines a set of system operations that are triggered over time by a set of workload events. A resource platform is a container for resources, i.e. hardware/software execution platform, that are used by the system operations mentioned above.

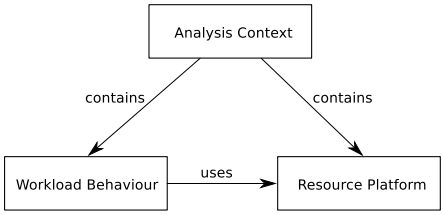


Figure 2.8: Structure of the MARTE analysis packages

2.2.4 SPACE

SPACE is a model-based engineering method [2] supported by the Arctis tool-set [42]. When this method is used, applications are composed of *building blocks* that can specify local behaviour as well as the interaction between several distributed entities. This specification style enables a rapid application development since, on average, more than 70% of a system specification comes from reusable building blocks provided in domain-specific libraries [43]. In turn, this strategy helps to reduce the expertise required in developing cross-domain applications. An additional benefit is the formal semantics of the specification defined by Kraemer and Herrmann [44], which makes it possible to verify system properties, e.g. that the building blocks are correctly integrated into activities [42].

Figure 2.9 gives an overview of the SPACE method [44]. An engineer starts studying a library of reusable building blocks. In case a needed building block does not exist in the library, an engineer can start creating a new one and add it into the library for its further reuse. Each building block can cover the behaviour of a single component as well as collaborative behaviour among several components. Building blocks can be domain-specific or quite general that can be integrated into several systems. Each building block is described as a combination of UML collaborations (an element of a UML composite structure diagram), activities (an element of a UML activity diagram), and so-called external state machines (ESMs) that specify externally visible behaviour of building blocks. Several building blocks are composed into a system with desired services. At this stage, analysis of a composed system (e.g. verification of functional or safety properties) can be performed due to the defined transformation of collaborative models into a temporal logic formula that serves as an input to the TLA (Temporal Logic of Actions) model checker [45]. Thereafter, the resulted system design is automatically transformed into state machines, that can be further used to generate implementation code via relevant transformations.

Our contribution enhances the step *composition and analysis* from Figure 2.9 when it comes to decide on a set of security measures expressed as reusable building blocks. In particular, we elaborate a method to select a set of security building blocks that are suitable for a system under development according to identified security needs.

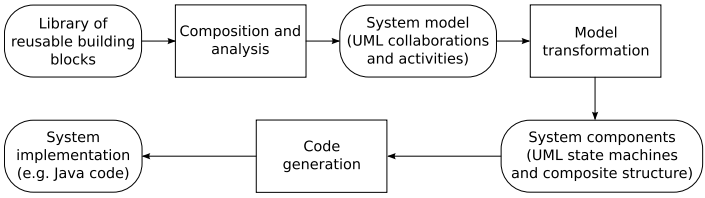
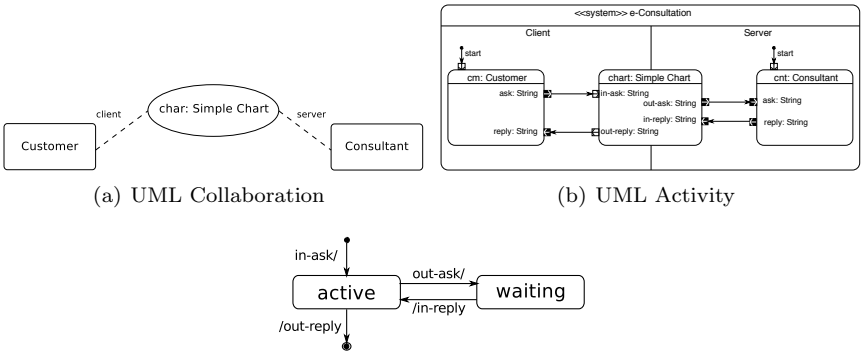


Figure 2.9: SPACE model-based engineering method, adapted from [2]

In the following, we explain elements of the modelling language used by SPACE, namely local blocks, collaborative blocks, and external state machines. We use a small example of a simple e-consultation application depicted in Figure 2.10 to demonstrate the introduced elements. In this scenario, a customer sends a question to a consultant. The consultant processes the question and sends a reply to the customer. The system structure is specified by a UML collaboration as shown in Figure 2.10(a). On this diagram, the collaboration roles depicted as rectangles represent two components of the system, namely a *Customer* and a *Consultant*. These two components are bound to the *client* and *server* roles respectively. The collaboration use, namely the *chart:Simple Chart* block, that is depicted as an ellipse encapsulates a logic of the component interaction.

Figure 2.10(b) shows the behaviour view of the system that is modelled as a UML activity with a slightly modified syntax. The e-consultation scenario is built of two partitions, i.e. the client and the server, that model the corresponding entities, i.e. a customer and a consultant. These partitions include three building blocks (instantiated as call behaviour actions), namely *cm:Customer*, *cnt:Consultant*, and *chart:Simple Chart*. The former two blocks model the local behaviour and are called *local blocks*, while the latter block models interaction between entities and called *collaborative block*. Each of these blocks is associated with another UML activity that details their behaviour (not shown in Figure 2.10).



(a) UML Collaboration

(b) UML Activity

(c) External State Machine

Figure 2.10: Model of a simple e-consultation application in SPACE

The overall activity is called *system block*. In our example, the local blocks are initiated with a special node denoted as filled circle (•). Pins at sides of building blocks are used to control their behaviour passing tokens of control or data flows along corresponding edges. The white pins represent pins that are used to start (the *start* and *in-ask* pins) or to terminate (the *out-reply* pins) building blocks. The dark pins denote streaming pins, i.e. pins that are used just to pass data objects. In our case, they are *ask*, *reply*, *out-ask*, and *in-reply*. The pins *out-reply* and *in-ask* transmit data objects as well, but these are activating and deactivating pins respectively, and, therefore, are coloured in white.

Figure 2.10(c) illustrates the ESM for the *Simple Chart* building block that is a modified UML state machine. The labels of the transitions refer to pins that sit on sides of the corresponding building block used to pass tokens. Thus, pins are used to activate transitions. The slash symbol (/) indicates if a transition is activated by an input (the slash symbol follows the label) or output (the slash symbol preceeds the label) pin.

Similar to functional building blocks, security mechanisms can be expressed as self-contained building blocks. SPACE has been already used

for encapsulating security functionality in the form of building blocks [46] validating their correct integration [47]. Additionally, the recent work of Gunawan and Herrmann [48] enables compositional verification of security properties for SPACE models.

2.2.5 Tools

MBE promotes the use of modelling for a set of sophisticated tasks of system development. It defines techniques to manipulate produced models, e.g. for simulation, verification, and transformation. These techniques, in turn, must be supported with corresponding tools to enjoy all benefits that are provided by MBE. This section outlines some basic and widely spread tools that enable the practices of MBE.

There are two main languages for meta-modelling, namely MOF (mentioned in Section 2.2.1) and Ecore. Recall, that a meta-model or abstract syntax defines the structure of a modelling language, i.e. its constructs, relations, and properties. Ecore is a meta-modelling language used within the Eclipse Modelling Framework (EMF) [49]. The EMF project is a widely used modelling framework that allows engineers to work with modelling languages. EMF provides facilities to define abstract and concrete syntax, and to create editors for custom models and Java code for developed meta-models. OMG defines two variants for MOF, namely Essential MOF (EMOF) and Complete MOF (CMOF). CMOF extends EMOF with additional structures. To specify a modelling language using EMOF, an engineer can use the Kermeta tool [50]. Alternatively, KM3 (Kernel Meta-Meta-Model) [51] is a textual language to create meta-models for DSMLs. Meta-models specified in Ecore or MOF can be serialised to an XMI file.

A variety of tools exist to define a concrete syntax for a DSML. For example, Graphiti [52] and GMF [53] are Eclipse-based graphic frameworks that allow developing custom editors. These tools enable automatic generation of a basic editor that can be further refined and tuned. Alternatively, an engineer can define the text representation of a concrete syntax for a modelling language using such tools as Xtext [54]. It provides a language to define grammars and a generator to create parsers and Eclipse-based editors for DSMLs.

A lot of tools are available for modelling with UML: MagicDraw [13], Enterprise Architecture [55], Rhapsody [56], to name some main examples. Additionally, EMF provides its own UML2Tool plug-in [57] for defining UML models. Most of the tools mentioned above already support the use of the MARTE profile providing corresponding plug-ins. Moreover, MARTE is implemented in the Eclipse-based Papyrus tool [58].

In our work, we use the MagicDraw tool together with its MARTE plug-in (in Chapters 4 – 5) since MagicDraw was used in a European project in which we participated. Besides, we use the Eclipse-based Arctis tool-set [42] to work with the language of the model-based engineering method SPACE

describe in Section 2.2.4.

Atlas Transformation Language (ATL) and Query/View/Transform (QVT) are M2M transformation languages. QVT (QVT Operational) [59] is an imperative language standardised by OMG that allows specifying unidirectional transformations. ATL [60] is a declarative and imperative (hybrid) language developed within EMF. To create transformations using these languages, an engineer needs to write a script. Henshin [61] and EMorF [62] are declarative EMF M2M transformation languages where transformations are specified graphically. There is also an extensive support for M2T transformations. For example, EMF provides Java Emitter Template (JET), Acceleo, and Xpand template-based languages. In this thesis, we use the Acceleo tool for transforming UML models into the OWL syntax.

2.3 Ontology Technologies

An ontology [63] represents knowledge in a particular domain as a set of concepts and their relations. This knowledge is formalised as a logic-based system and described by knowledge representation languages. In particular, we use the Web Ontology Language (OWL2) [64] which is a commonly used and standardised language for creation of large ontologies.

OWL represents an ontology as a sequence of axioms. These axioms describe *classes*, *relations* between classes, and their *individuals*. An OWL *class* declares the concept of a domain and can be refined by sub-classes. OWL *individuals* are instances of OWL classes. OWL supports two types of relations. OWL *object property* defines a relation between two individuals, where one of them plays the role of a *domain*, and another one plays the role of a value *range*. In other words, *domain* defines a subject of a relation, whereas *range* defines an object. OWL *datatype property* serves to introduce relations between an individual (domain) and the XML schema datatypes (range) known as XSD (XML Schema Definition). XSD provides such primitive data types as *boolean*, *integer*, etc.

The OWL language supports a set of constructs that facilitate management of ontologies. In particular, the OWL language implements the importing feature, which allows relating different OWL ontologies using the *owl:import* statement. When merging two or more ontologies, it may be the case that these ontologies contain overlapping concepts that have different names, but actually refer to the same things from the reality. Such similar concepts should be related in the merged ontology using the construct *owl:sameAs*. This procedure sometimes is referred to as ontologies alignment that is the process of determining correspondences between concepts.

OWL ontologies enable querying of the declared knowledge by combining ontology reasoners (e.g. Pellet or HermiT) and SPARQL querying language [65]. SPARQL 1.1 is a standard query language (recommended by W3C) to execute data queries on top of OWL. It supports yes/no-questions (the *ASK* query form), a selection which matches a desired pattern (the *SE-*

LECT query form), filtering (the *FILTER* modifier), sorting (the *ORDER* modifier), string matching, etc. In this thesis, we use SPARQL together with HermiT.

To design and manage an ontology, one can use such a tool as Protégé [66]. Since tools developed in this work are Java-based, we exploit the Java OWL API [67, 68] to manipulate ontologies (i.e. addition and modification of axioms). To execute SPARQL queries, one can load an ontology into the Protégé tool and use its SPARQL plug-in [69]. In our work, we use Java APIs provided by the widely accepted Jena [70] framework to query ontologies. It provides the SPARQL compliant query engine (among other services) for OWL ontologies.

The OWL standard [71] defines three variants of OWL, namely OWL Lite, OWL DL (Description Logic), and OWL Full. These three sublanguages have different expressiveness and, consequently, different complexity, and, therefore, are used for different purposes. OWL Lite is the simplest variant and allows the user to capture classification hierarchy and constraints with restricted expressiveness. For example, it permits only 0 and 1 as cardinality values. However, it has a simple implementation and comparatively easy to use. In contrast, OWL DL is the most expressive variant. It supports all OWL constructs, but their use is restricted by a set of requirements and rules outlined by W3C [71]. These constraints maintain computational completeness and decidability of this language. OWL Full relaxes constraints of OWL DL and enjoys all capabilities of OWL constructs. The price for this freedom is absence of any computational guarantees.

Ontologies for Security

A number of ontologies for security have been proposed. For example, Herzog et al. [72] and Fenz and Ekelhart [73] introduce two ontologies that formalise the domain of information security from different aspects; Kim et al. [74] present an ontology for annotating web-services; Karyda et al. [75] propose an ontology to assist reuse of the experts' security knowledge in the area of e-government applications. In our work, we adopt the ontology presented by Herzog et al. since it is built upon classic components of risk analysis. We continue with a brief description of this ontology.

The core of the Herzog et al. ontology [72] consists of six classes. Four of them are concepts related to risk analysis, i.e. asset, vulnerability, threat, and countermeasure. The remaining two classes are security goal and defence strategy. Relations between these concepts are defined as follows: an asset can *have* several vulnerabilities; a threat *threatens* assets with respect to some security goals; a countermeasure *protects* assets with respect to security goals by means of defence strategies.

The ontology gives diverse classifications of countermeasures, assets, threats, and vulnerabilities relevant for information security. In particular, the ontology defines 133 countermeasures, 79 assets, 88 threats, and 14 vulnerabilities. The security goal and defence strategy classes are described

by a set of individuals. Six individuals are defined for the defence strategy class, namely correction, deflection, detection, deterrence, prevention, and recovery. Fifteen individuals are defined for the security goal class, e.g. confidentiality, integrity, authorisation, and anonymity.

2.4 Semi-Markov Chains

A semi-Markov chain (SMC) [76] is a general state transition system defined by three components (S, P, H) where S is a set of states, $P = (p_{ij})$ is a transition matrix, and $H = h_{ij}(.)$ is a holding time distribution matrix. Each p_{ij} is the probability that a SMC that enters state i on its last transition will enter state j on its next transition. Each p_{ij} satisfies the standard conditions:

$$p_{ij} \geq 0, i = 1, \ldots, N; j = 1, \ldots, N, \sum_{j=1}^{N} p_{ij} = 1$$

N is the number of system states. Each $h_{ij}(.)$ is a holding time mass function of the process in state i when the next transition is to state j.

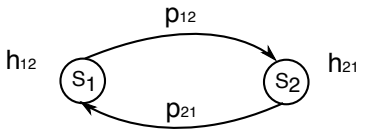


Figure 2.11: An example of a semi-Markov chain

A SMC depicted in Figure 2.11 has two states s_1 and s_2. The SMC moves to state s_2 from s_1 with the transition probability p_{12} and to state s_1 from s_2 with the transition probability p_{21}. However, before making a transition the process holds for some time τ in a current state. This holding time for each state is given by assigning a probability mass function that expresses the time that a system will stay in a state before proceeding to a next state as exemplified in Figure 2.11. In our example, $h_{12}(t)$ and $h_{21}(t)$ are holding times in states s_1 and s_2 respectively. Note that in general case holding time can depend on the next transition, therefore, the notation for holding times contains both the initial state and the destination. Hence, $h_{12}(t)$ is interpreted as holding time for a process in state s_1 given that the next transition is taken into state s_2. Thus,

$$P(\tau_{ij} = t) = h_{ij}(t) \tag{2.1}$$

We consider a case where holding time distributions are discrete distributions. There is also an assumption [76] that all holding times are at least one time unit in length, i.e. $h_{ij}(0) = 0$.

A SMC is characterised by $\phi_{ij}(n)$ that is called the interval transition probability from state i to state j in the interval $(0, n)$. The interval transition probability is defined by the following system equation:

$$\phi_{ij}(n) = \sigma_{ij}\ w_i^{>}(n) + \sum_{k=1}^{N} p_{ik} \sum_{m=0}^{n} h_{ik}(m)\phi_{kj}(n-m) \tag{2.2}$$

In equation (2.2), $\sigma_{ij} = 1$ if $i = j$, otherwise, $\sigma_{ij} = 0$; $w_i^{>}(n)$ is the probability that a SMC leaves its state i at a time greater than n. The second part of equation (2.2) represents the probability that a SMC will enter state j from i sequentially passing some states k before some time m and then continues along any path from k to j in the remaining time $n-m$.

2.5 Metering Infrastructure

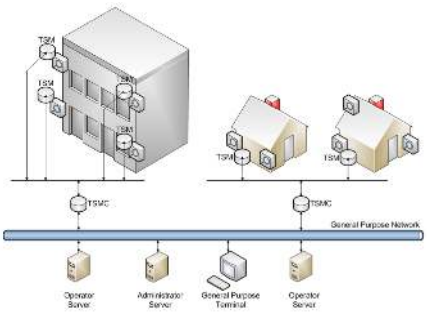


Figure 2.12: Overview of the smart metering infrastructure [3]

Figure 2.12 gives an overview of an infrastructure called Trusted Sensor Network (TSN) from the smart metering domain. This case is provided by the MixedMode company that has participated in the European SecFutur project [17]. TSN is built of a set of metering devices, database servers, client applications, and a communication infrastructure. The main goal of this system is to measure energy consumption at households and to associate measurements with the clients' data for billing purposes.

The actual measurement is done by Trusted Sensor Modules (TSMs) consisting of a computing platform and physical sensors. These devices can be distributed in households or office buildings. The acquired measurement data is transferred via a local bus from each TSM to a Trusted Sensor Module Collector (TSMC). All measurements collected by TSMCs are eventually sent to operator servers through a general-purpose network. Administrator servers and terminals can also be connected to this network for maintaining TSMs and TSMCs. The TSMC is also an embedded device, similar to TSM but with more functionality. That is, TSMC and TSM are two functional modules that are implemented on the same physical platform.

The overall specification of this case consists of 7 main scenarios that have a range of diverse security considerations. In this thesis, we focus on the measurement data transfer from TSM to TSMC and from TSMC to an operator server. Consequently, we concentrate on those security issues that concern confidentiality and integrity of the measurement data produced, collected, or stored by the system components.

3

SEED: Bird's Eye View

This chapter presents a **S**ecurity-**E**nhanced **E**mbedded system **D**esign (SEED) approach, that provides concepts, methods, and tools for dealing with security issues of embedded systems already at the design phase.

3.1 Introduction to SEED

SEED addresses the situation when system engineers are not necessarily security experts, and when security experts are not easily accessible to assist system engineers. Overall, the SEED approach rests on three basic principles discussed in the introduction:

- model-orientation,
- domain specialisation, and
- separation of responsibilities and concerns.

Besides these principles, there is another significant design consideration, namely the proposed approach is defined as an increment to existing practices and process models known for embedded system development. These concerns for security-related processes have been revealed by Whyte and Harrison [77]. The authors point out that among factors that prevent enforcing security in a systematic way are a lack of security expertise and hesitation of system engineers to commit to follow a new approach that deals with the security aspects due to associated risks.

The SEED approach can be considered in two parts that describe it at two levels of abstraction. These levels are SEED *foundation* and SEED *realisation*. The SEED *foundation* is a generic form of the proposed process that can be instantiated for different technologies, modelling and formal languages. The SEED foundation defines main activities, involved roles, used principles, and their exploitation. The SEED *realisation* is an implementation of the generic process on a selected set of technologies and languages. In this thesis, the set of languages and technologies selected for one SEED realisation are SPACE, MARTE, and ontology.

This chapter focuses on presenting the generic process, i.e. the SEED foundation. The realisation details are explained in Chapters 4 – 6 where we also illustrate application of the SEED using scenarios from the smart metering infrastructure.

3.2 The SEED Foundation

Figure 3.1 depicts the generic process[1] of the SEED foundation. It consists of three activities: *creation of a system model*, *capturing of the domain-specific security knowledge*, and *development of a security-enhanced embedded system*. These activities are performed by embedded system engineers or security experts. Thus, the SEED approach acts as a vehicle for communication between the two expert groups. We continue explaining each of the above-mentioned activities.

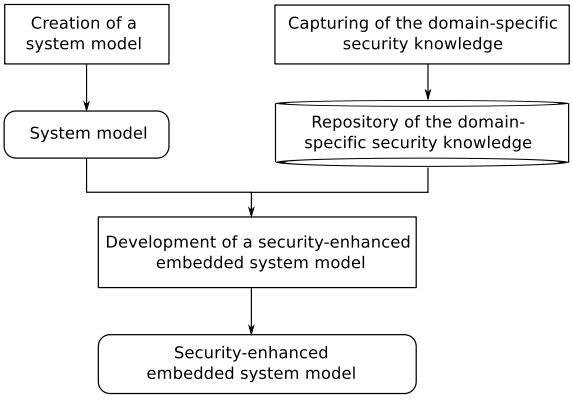


Figure 3.1: Generic process – the SEED foundation

[1]It has been developed in cooperation with research partners from the EU FP7 SecFutur project [17].

3.2.1 Creation of a System Model

The activity of creation of a system model is performed by an embedded system engineer. Figure 3.2(a) depicts a fragment of a simplified embedded system design flow that demonstrates artefacts that are used for the SEED foundation. Note, that this figure does not detail how an embedded system engineer produces the artefacts (see the process models described in Section 2.1).

An embedded system engineer starts from system specifications and creates a functional model of a system and a set of models for potential execution platforms. A functional model of a system is a set of modelling elements that describe structural and behavioural aspects of an application, e.g. UML class and activity diagrams respectively. An execution platform model of a system describes an assembly of resources. Each resource provides some services to support execution of an application described as a functional model. Thereafter, an embedded system engineer proceeds to select the best-fitted execution platform. This step includes allocation of elements of a functional model onto available resources, i.e. onto the execution platform model. In other words, allocation is used to establish an association between elements of functional and execution platform models. To sum up, the generic process, i.e. the SEED foundation, uses three artefacts produced while designing an embedded system. These artefacts are a functional model, execution platform model (or just platform model), and allocation information.

Hence, SEED is intended to support security analysis when a system engineer has a version of a system design when some decisions about functionality of an embedded system and its hardware/software architecture have been made. At this point, an embedded system engineer has relevant information needed to estimate the system's capabilities to deal with security aspects. In particular, the initial system design is present, valuable objects and actions that influence them are known, and the capacity of a system dedicated for security enforcement of security can be indicated.

3.2.2 Capturing the Domain-specific Security Knowledge

Capturing of the domain-specific security knowledge is an essential step to enable further reuse. This activity is conducted by a domain security expert (just a security expert or a security engineer from now on), i.e. a security expert who has knowledge about an application domain. The main task that a security expert completes as part of this activity is to describe available security mechanisms. Additionally, a security expert provides other relevant information that is needed for an embedded system engineer to make informed decisions when selecting a suitable set of security mechanisms.

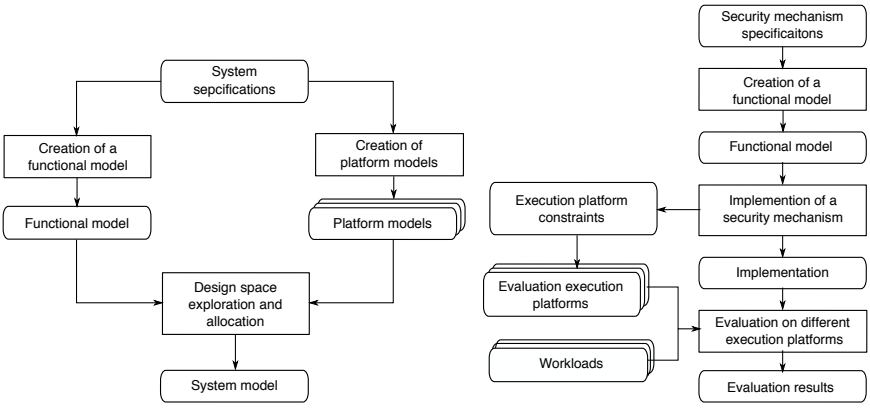
Figure 3.2(b) illustrates a simplified flow of a security mechanism development that demonstrates artefacts that are used for the SEED foundation.

First, a functional model is created based on some specifications of a security mechanism. For example, it can be a mathematical abstraction of a mechanism, e.g. an algorithm defined as a mathematical object. Similarly to an embedded system, a functional model defines behaviour and structural aspects of a security mechanism. Thereafter, a security expert proceeds to implement it. At the implementation phase, besides an implementation itself, a security expert produces constraints on execution platforms. These constraints come from design characteristics of a security mechanism. For example, a certain implementation of an encryption algorithm can rely on a certain instruction set that should be present in a micro controller. These constraints restrict a set of suitable evaluation execution platforms selected at the next step. Thereafter, the evaluation step allows studying performance aspects of a security mechanism (i.e. the created resource overhead for a provided security level) on a selected set of evaluation execution platforms given certain workloads. Ideally evaluation platforms should mimic candidate execution platforms where a security mechanism is expected to be deployed. For example, a security expert may consider different assemblies of systems or different configurations of the same system. A workload is a set of stimulus under which a security mechanism should be evaluated, e.g. a stream of input events. This also should resemble the expected use of a target system. The main intention of this evaluation is to study different aspects of security mechanisms and to collect a range of performance indicators that can be later used by a system engineer to reason about applicability of a security mechanism in the context of a certain system. This information about evaluation platform candidates and workloads comes from the application domain knowledge.

The set of artefacts mentioned above, namely a functional model, execution platform constraints, data about performance evaluation (evaluation platforms, workload, and results of evaluation) constitute the *domain-specific security knowledge* where a domain represents an application domain like smart metering devices and set-top boxes in the context of this work. Additionally, the domain-specific security knowledge includes a declaration of security properties provided by a security mechanism. All together these artefacts give a holistic view on existing security solutions needed to support their integration into an embedded system. For example, an embedded system engineer can study such aspects as: whether a system integrated with a security mechanism still maintains its dedicated functionality in a satisfactory way; or whether a candidate security mechanism fits in the resource-related constraints of an embedded system; or whether security properties of a security mechanism correspond to formulated security requirements of a system.

The domain-specific security knowledge is stored in a repository so that it is available for an embedded system engineer for its further reuse. The domain specialisation of security knowledge is motivated by the following rationale. Security requirements for a particular application depend on po-

tentially present threats that vary based on the nature of an application and deployment environment (i.e. a domain). As a result, the domain-specific security knowledge will have a different set of required security properties and, consequently, associated with them security mechanisms that satisfy these security properties.



(a) Embedded system (b) Security mechanism

Figure 3.2: Fragments of simplified design flows

3.2.3 Role of an Application Domain

The notion of an application domain plays an important role in SEED. It is a basic notion that creates a common ground for system engineers and security experts.

In general, a domain can be simply defined as an area of interest to a particular development [36]. Kelly and Tolvanen [36] mention two general ways to look at the notion of a domain: horizontal and vertical. Examples of horizontal (also called technical) domains are persistency, user interface, and communication; examples of vertical (also called as functional or business) domains are telecommunication, banking, and robot control. In this thesis, our notion of application domains has more in common with the vertical domains mentioned by Kelly and Tolvanen.

In domain-specific modelling the main objective of creating a domain is to define a domain-specific language that will facilitate development effort. In our work, we look at a domain as a pool that contains domain-specific information. In particular, this pool contains information about components that are used in this domain for construction of execution platforms, and information about workloads that are typical for this domain. These details can be encapsulated as a part of a domain-specific language.

Thus, when a system engineer designs an execution platform he/she uses

constraints and components from the corresponding domain pool; similarly, a security experts relies on the same common components and constraints when designing a security mechanism. Additionally, when a security expert evaluates a security mechanism he/she does it based on workloads that are typical for systems from this domain. Figure 3.3 summarises the relation between a domain, a system, and a security mechanism. In this fashion, we can achieve some degree of relative independence between security experts and system engineers.

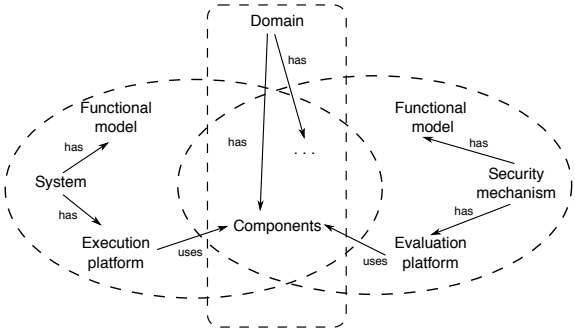


Figure 3.3: Partial relations between a domain, a system, and a security mechanism

3.2.4 Development of a Security-enhanced Embedded System

An embedded system engineer is the one who develops a security-enhanced embedded system. The goal is to extend a system model with security features that are retrieved from relevant parts of the domain-specific security knowledge. In particular, this activity within SEED foundation is built of three basic steps:

- First, a system model is analysed to identify those parts that need security protection. Both functional and execution platform models are subjects of this analysis.
- Second, the security knowledge is consulted to retrieve a set of relevant security properties (based on outcomes of the previous step). Consequently, a set of security mechanisms that are available in the repository to satisfy identified security properties are also retrieved from the domain-specific security knowledge.
- Finally, since integration of a new feature into an embedded system comes with resource claims, the selected set of security mechanisms is studied with respect to a potentially created resource overhead.

The result of this activity (if successful) is an embedded system design extended with a suitable set of security mechanisms that meet security goals of a system under development.

In this section, we have described the SEED foundation that presents the generic process intended to support an embedded system engineer and security expert in designing security-enhanced systems. In the next chapters, we describe one instance of the SEED realisation. In particular, we clarify in terms of specific concepts, methods, languages, and technologies how the domain-specific security knowledge is captured by a security expert (in Chapter 4) and how it is applied by an embedded system engineer (in Chapter 5).

4

Capturing of the Domain-specific Security Knowledge

This chapter details the "capturing of the domain-specific security knowledge" activity for the SEED realisation. Recall from Figure 3.1, that as part of this activity, a security expert creates a set of artefacts to describe the existing security mechanisms. This description includes functional model of security solutions, their provided security properties, and information about their performance evaluation. In order to structure and operate with this information, we develop two concepts, namely a Domain-Specific Security Model (DSSM) and a Performance Evaluation Record (PER). These concepts are formalised as a set of ontologies. Additionally, we employ methods and tools from the area of model-based engineering, like modelling languages and transformation techniques, to facilitate the creation and usage of these concepts by embedded system engineers and security experts.

We describe the developed concepts and their support in Section 4.1. Thereafter, Section 4.2 explains processes and tools created to assist a security expert to work with the introduced concepts.

4.1 Developed Concepts and Artefacts

As mentioned above, our approach rests on two concepts. DSSM is a concept used by a security expert to describe a security solution; PER is another one that serves to describe results of performance evaluation of a security solution. These concepts are implemented by aligning two technologies, namely UML modelling and ontologies. In the following, Section 4.1.1 and

Section 4.1.2 explain the DSSM and PER concepts respectively.

4.1.1 Domain-specific Security Model

A scheme depicted in Figure 4.1 shows the introduced artefacts and their relations. The core artefact is an ontology that we use to define the structure for description of the domain-specific security knowledge. Among studied ontologies for the security domain, the one created by Herzog et al. [72] fits our needs. This ontology is a general information security ontology and was explained in Section 2.3. The ontology created in our work is an adapted Herzog et al. security ontology and is called *core security ontology*. To facilitate the use of this ontology we define an expert's front-end using the widely accepted UML standard. In particular, the core security ontology is represented as a UML class model. This UML class model serves as a simple tool used by security experts to describe their knowledge about existing security solutions. More specifically, an instance of this model, i.e. object diagram, is actually the captured domain-specific security knowledge and is called Domain-Specific Security Model (DSSM). We transform each DSSM into the OWL syntax and use it to extend the original core security ontology. We refer to an extended ontology as an *enriched security ontology*. In the following, we continue explaining the mentioned artefacts in more detail.

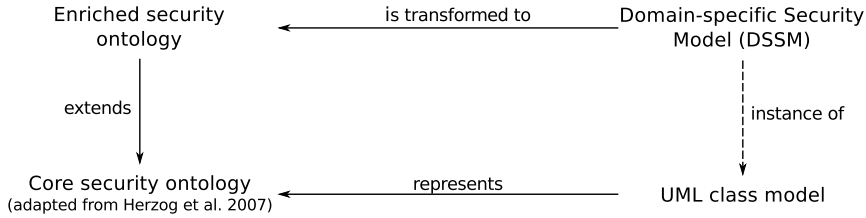


Figure 4.1: DSSM concept and related artefacts

Core Security Ontology

The core security ontology adapted from the Herzog et al. [72] ontology is depicted in Figure 4.2. The main point of departure arises in order to introduce three new concepts, which are *security property*, *security building block*, and *domain*. Thus, the use of the Herzog et al. ontology has served its purposes outlined by the authors [72], namely as a learning material about the structure of information security and as a framework for developing new detailed security taxonomies. We proceed to describe the adapted ontology used to support the processes of capturing and using the domain-specific security knowledge.

In the core security ontology, we reuse three basic concepts introduced by Herzog et al. [72], namely *asset*, *security goal*, and *defence strategy*. Assets are the "objects of value" of a system that are needed to be protected. In

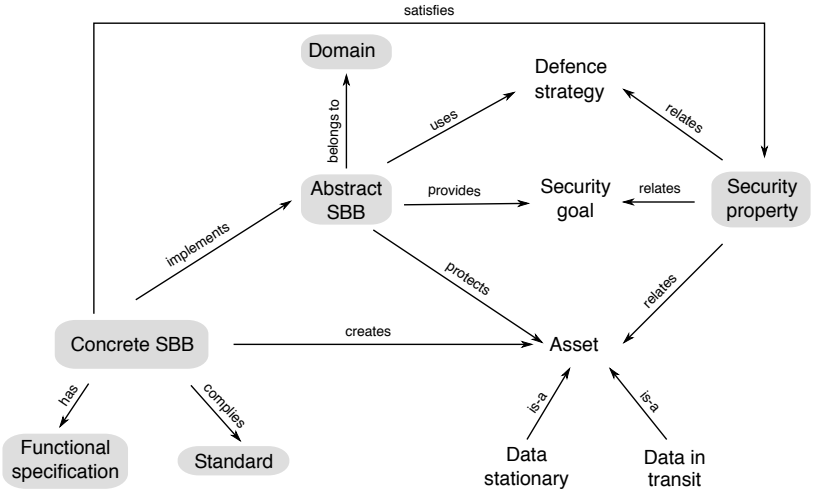


Figure 4.2: Illustration of the core security ontology

our context, they can be *stationary data* residing on a physical component or *data in transit* being transmitted between different components. Other types of assets, e.g. algorithms or Intellectual Properties (IPs), may also be considered and introduced. The protection of an asset leads to the fulfilment of a particular security goal like protecting its confidentiality, integrity, or availability. The countermeasures introduced below follow a certain defence strategy, e.g. preventing attacks or recovering after an attack. For the security goal and for the defence strategy, we reuse all the terms (i.e. individuals) defined in the ontology of Herzog et al. [72].

In addition to these elements, the core security ontology introduces new concepts, which are shown as grey boxes in Figure 4.2. Two of them are *abstract* and *concrete Security Building Blocks* (SBBs) replacing the notion of a countermeasure used by Herzog et al. [72]. These refinements enable us to distinguish between more general countermeasures represented by the abstract SBBs and their implementations specified as concrete SBBs. For example, an abstract SBB might refer to a cryptographic hash function as a general method to provide integrity while the different realisations of the hash function (e.g. SHA-1, MD2, or MD5), that are implemented as a piece of code or hardware, are represented by concrete SBBs. With respect to the resource limits of embedded systems, it is important to note that the implementations may have different resource footprints. Each concrete SBB is associated with *functional specifications*. The functional specification entity is description of a functionality of a concrete SBB (that is an implementation of a security countermeasure) to the extent (level of details) that is needed for integration of this SBB into a system model. A particular form of these descriptions will dependent on modelling language of this domain. For ex-

ample, in case of the component-based system development, the functional specifications of a SBB can be a specification of a corresponding component and its interfaces. In our work, we use the SPACE building blocks that are composed of a functional model and external state machine. Further, a concrete SBB can comply with some standards, e.g. it may have passed some certification. Another concept introduced by our ontology is the notion of *security properties* encompassing the three notions: assets, security goals, and defence strategies. Finally, we enrich the ontology with the concept of a *domain* that represents an application domain, e.g. the smart metering domain.

The relations in the core security ontology are defined as follows. Like the countermeasures in the Herzog et al. ontology, an abstract SBB *protects* an asset, *provides* some security goals, and *uses* some defence strategies. We have modified the *protects* relation for a security goal and a defence strategy used by Herzog et al. [72] since we find that the *provides* and *uses* relations reflect more precisely their semantics within our process. In addition to these three relations, an abstract SBB *belongs to* some application domain. A concrete SBB *implements* an abstract SBB but, in turn, may *create* certain assets itself. For example, the keys in some implementation of a public key cryptography mechanism have to be protected in order to fulfil the confidentiality and integrity goals. The functional specification concept is related to the concrete SBB concept with the *has* relation. The last relation of the concrete SBB concepts is a *comply* that relates it to the standard concept. This covers common requirements in engineering of networked embedded systems. In the metering domain, for example, a system will have to fulfil legal calibration requirements following a standard. Finally, a security property *relates* assets, security goals, and defence strategies, which in the following sections will be referred to by triplets `[asset, security goal, defence strategy]`.

The UML Representation

To support a security expert in capturing the domain-specific security knowledge that is formalised as the core security ontology, we represent the ontology as a UML class model depicted in Figure 4.3. Each DSSM is effectively an instantiation of the core security ontology. Therefore, we specify a DSSM as an instance of this class model. The security knowledge captured by each DSSM is used to extend our core security ontology presented above with a corresponding set of axioms on relations and individuals. This enables us to use the ontology querying and reasoning services to extract relevant parts when this knowledge is required for an embedded system engineer. In other words, the class model in Figure 4.3 serves as a language dedicated for capturing knowledge by security experts, i.e. to create DSSMs, while the ontology in Figure 4.2 is a formalism for this language [36].

It is worth mentioning, that the UML model in Figure 4.3 is not a direct transformation of the security ontology from Figure 4.2 since it is not

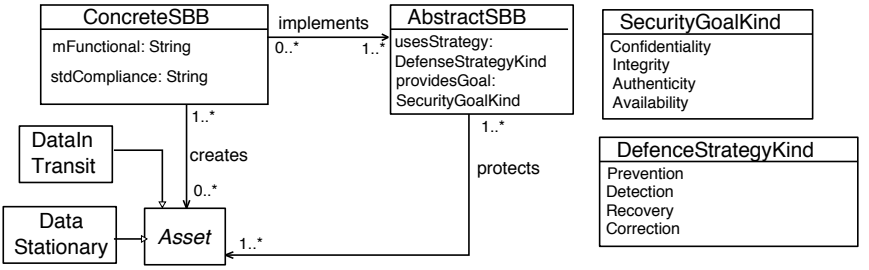


Figure 4.3: UML representation of the core security ontology

domain knowledge by itself, but rather a tool to capture the knowledge. The model consists of five classes and three relations, which are direct mappings of the elements of the core security ontology. The preserved classes are *Asset*, *AbstractSBB*, *ConcreteSBB*, *DataStationary*, and *DataInTransit*. The preserved relations are *implements* (between *ConcreteSBB* and *AbstractSBB*), *protects* (between *Asset* and *AbstractSBB*), and *creates* (between *ConcreteSBB* and *Asset*). The *is-a* relation from *DataStationary* and *DataInTransit* to *Asset* in the core security ontology is modelled in the form of generalisations.

Other elements of the core security ontology are specified in a different way. Instances of the security goal and defence strategy classes are represented as the enumerations *SecurityGoalKind* respective *DefenceStrategyKind*. The *uses* relation between the abstract SBB and defence strategy classes in our ontology is represented by the property *usesStrategy* in the class *AbstractSBB*. Likewise, the *providesGoal* property in *AbstractSBB* replaces the *provides* relation between the abstract SBB and security goal classes of the core security ontology. Similarly, the functional specification and standard concepts related to the concrete SBB concept are represented as the *mFunctional* and *stdCompliance* properties of the *ConcreteSBB* class respectively.

In contrast, the security property and the domain concepts of the core security ontology are not directly represented in the UML model. This is due to the fact that the triple asset, security goal, and defence strategy already captures the notion of a security property. Therefore, security properties can be directly extracted from a DSSM. Analogously, the domain concept from the core security ontology is represented by the name of a DSSM (i.e. the object diagram).

As mentioned before, a given DSSM is essentially an instance of the UML class model depicted in Figure 4.3. As an example, we depict in Figure 4.4 the UML class diagram for a small fragment of the metering DSSM used for our measurement transfer scenario from Section 2.5. This DSSM contains three assets: *StoredMeasurement* that represents energy measurements stored on a device; *CollectorToServerMsr* that represents energy measure-

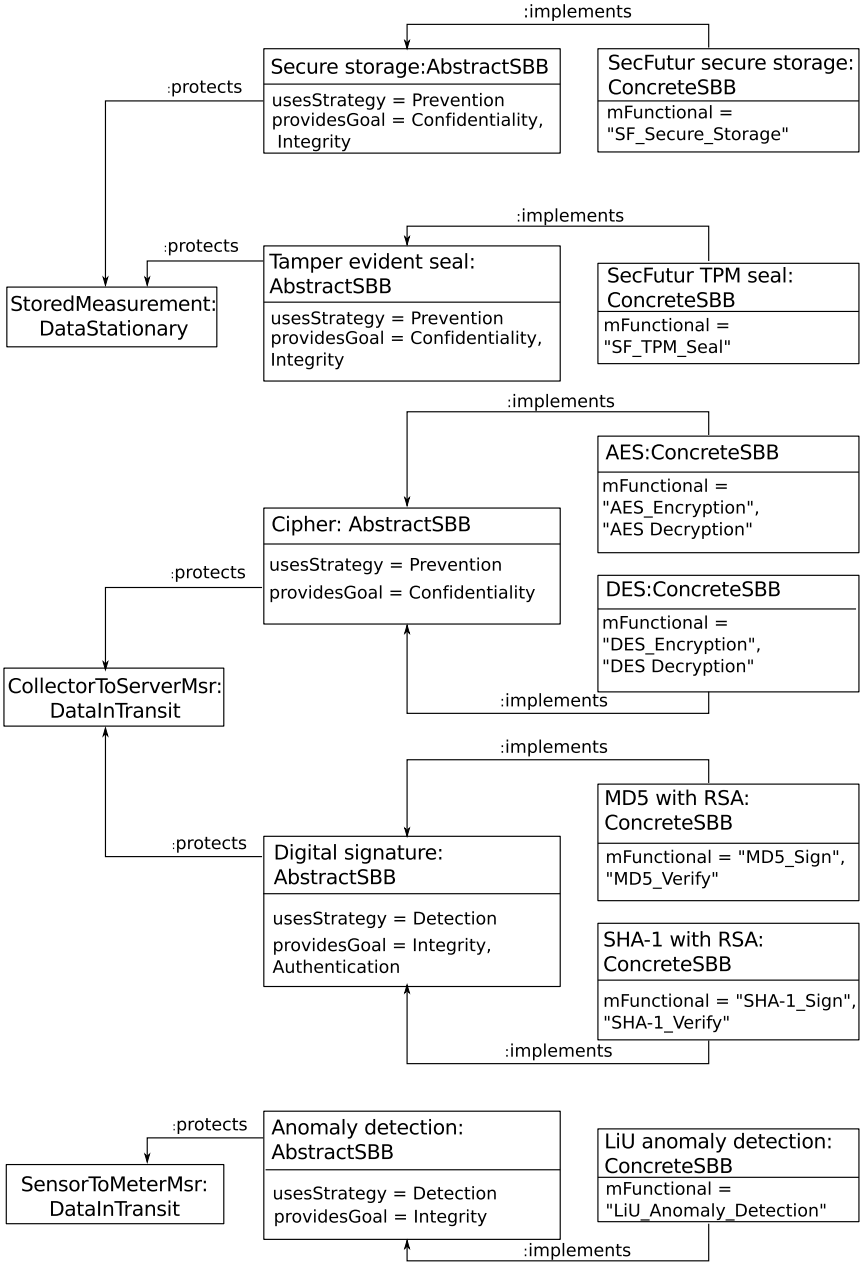


Figure 4.4: Metering DSSM

ments sent from a collector device to an operator server; *SensorToMeterMsr* that represents energy measurements sent from a sensor to a metering de-

vice. These assets may be protected by five abstract SBBs: *Secure storage* and *Tamper evident seal* that provide confidentiality and integrity for the *StoredMeasurement* asset; *Cipher* that provides confidentiality for the *CollectorToServerMsr* asset; *Digital signature* that provides integrity and authentication for the *CollectorToServerMsr* asset; and *Anomaly detection* that provides integrity for the *SensorToMeterMsr* asset. Seven concrete SBBs implement these abstract SBBs. Each abstract SBB in Figure 4.4 is supported by one or two implementations. In general, one abstract SBB can be implemented by several concrete SBBs. For example, the *DES* and *AES* concrete SBBs implement the *Cipher* abstract SBB. These concrete SBBs have one or a pair of functional models (see the mFunctional slot). In this study and in this SEED realisation, they are SPACE blocks for encryption (*AES_Encryption* and *DES_Encryption* respectively) and decryption (*AES_Decryption* and *DES_Decryption* respectively).

Enriched Security Ontology

In the following, we explain the last artefact introduced in Figure 4.1, namely the enriched security ontology.

Each DSSM is transformed into the OWL syntax (outlined in Section 2.3) extending the core security ontology, as it is depicted in Figure 4.1. In other words, these DSSMs are used to populate the core security ontology. In turn, each DSSM is encapsulated into a separate ontology, called *[domain name] security ontology* that imports (using the *owl:import* construct) the core security ontology. We refer to the merge of all domain security ontologies obtained from DSSMs as the *enriched security ontology*. This modular structure of the enriched security ontology facilitates its management (e.g. its update). Figure 4.5 shows the described dependencies of the introduced ontologies. In this figure, *Domain 1* and *Domain n* represent application domains, e.g. metering devices and set-top boxes domains respectively.

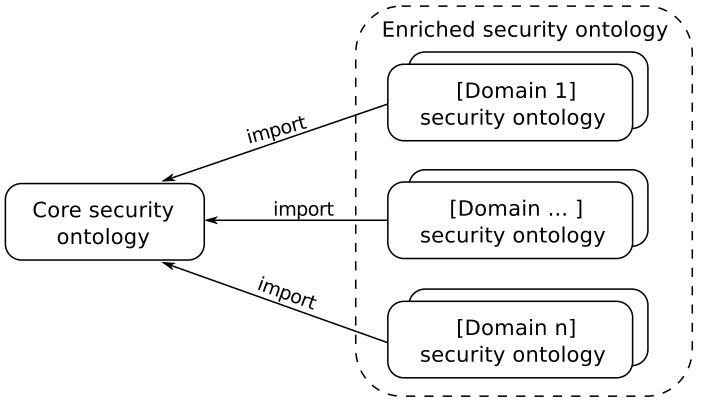


Figure 4.5: Enriched security ontology

The task of updating the enriched security ontology with the knowledge captured by a newly created DSSM is implemented as transformation of elements of a DSSM and their relations into corresponding set of axioms on classes, relations, and individuals. We use model-to-text transformation techniques and tools, like Acceleo [78], to realise this transformation. Afterwards, these axioms are added into the corresponding *[domain name] security ontology*.

All objects of the metering DSSM in Figure 4.4 are added into the enriched security ontology as individuals of the corresponding concepts of the core security ontology from Figure 4.2. Thereafter, additional axioms are added that transform instances of the *protects*, *implements*, and *creates* associations from a DSSM into corresponding (OWL) object properties. The *usesStrategy* and *providesGoal* attributes are transformed into triples of `[object, object property, subject]` where an object is an instance name of the AbstractSBB class, an (OWL) object property corresponds to the *uses* and *provides* relations respectively, and a subject is a value of the attribute. Such attributes as *mFunctional* and *stdCompliance* are approached in a similar way. The difference is that an (OWL) object property is replaced by a (OWL) data property since values of these three attributes are strings.

4.1.2 Performance Evaluation Record

Figure 4.6 depicts a scheme that shows the defined artefacts and their relations. This scheme follows a similar pattern as presented in the previous section. The core artefact is an ontology that describes concepts and relations of the performance evaluation domain. This ontology is called *core evaluation ontology*. Differently from the core security ontology, a UML profile is designed as a representation front-end for the core evaluation ontology. This profile is called Generic Evaluation Model (GEM). An expert defines an evaluation context as a set of UML models of different types and annotates their elements using stereotypes of the designed profile. These UML models are called Performance Evaluation Record (PER). Further, these models are transformed into the OWL syntax and used to extend the initial core evaluation ontology forming an *enriched evaluation ontology*.

The infrastructure in Figure 4.6 can be used to capture results of performance evaluation of any building block that realises extra-functional properties, e.g. security, safety, and real-time. Thus, security is just one example of such extra-functional properties. Therefore, we use generic terms in this section, i.e. the SBB term introduced in the previous section is replaced by the *Reusable Building Block (RBB)* concept [1]. In the following, we continue explaining in more detail the mentioned artefacts.

[1] In the rest of the thesis (except this section), we avoid using the term RBB. However, a reader should understand that the term SBB should be replaced by RBB when it comes to discussions of PER and related concepts (e.g. in Section 4.2 and 5.5).

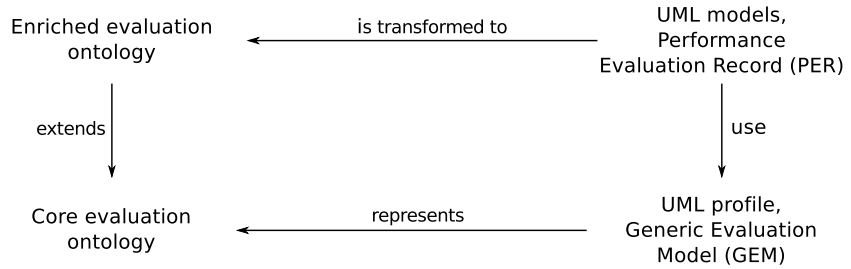


Figure 4.6: PER concept and related artefacts

Core Evaluation Ontology

Figure 4.7 depicts an ontology developed in this study in order to capture results of performance evaluation of RBBs called *core evaluation ontology*. To build the core evaluation ontology, we use the knowledge about the structure of the analysis domain obtained from various sources. They are mainly the MARTE GQAM profile explained in Section 2, and the published research and industrial papers where performance aspects of different RBBs are studied, e.g. such works published by Nadeem and Javed [79] and Preissig [80] where performance characteristics of security solutions are investigated. We proceed to describe elements of the core evaluation ontology depicted in Figure 4.7.

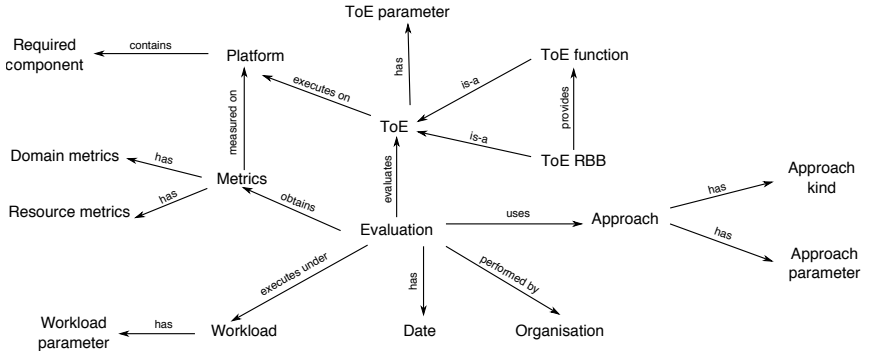


Figure 4.7: Core evaluation ontology

Evaluation is a central concept of the core evaluation ontology. It relates four other concepts that indicate constituents of any evaluation procedure. They are *Target of Evaluation* (ToE), *approach*, *workload*, and *metrics*.

ToE refers to an RBB that is under performance evaluation. There are two sub-classes for the ToE concept, namely *ToE RBB* and *ToE function*. A ToE can be characterised by a set of parameters introduced by the *ToE parameter* concepts, e.g. the key size for an encryption RBB. The *platform* concept denotes an evaluation platform used for performance analysis. We

create the *required component* concept to introduce those components of an evaluation platform that are significant for a considered ToE to obtain the captured performance results. For example, it can be a particular instruction set exploited by a ToE implementation.

The *approach* concept denotes the description of the used evaluation method. Each approach can be characterised by its *kind* and *approach parameters*. The *approach kind* concept defines the type of the evaluation method. It has such individuals as simulation, emulation, or analytical analysis. The *workload* concept introduces the workload used during the performance evaluation, e.g. a sequence of triggering events, or an amount of data sent over a channel. Similarly, a workload can be parameterised using the *workload parameter* concept.

The *metrics* concept defines a set of metrics adopted for the performance evaluation. These metrics can be of two categories as denoted by its sub-classes. The first category (the *resource metrics* concept) describes the resource footprint created by ToE (e.g. execution time). The second category (the *domain metrics* concept) refers to the obtained indicators that characterise the quality of extra-functional properties, e.g. the security level provided by a ToE. The presence of these two categories reflects the fact that for selection of suitable RBBs both resource footprint and quality of service indices play a significant role.

The following relations are defined in the core evaluation ontology. Any evaluation *evaluates* some ToE, *executes under* some workload, *uses* some approach, and *obtains* some metrics. A ToE RBB can *provide* some ToE functions. Any ToE *executes on* some evaluation platform. A platform may *contain* some required components. Any obtained metrics are *measured on* some platform. A ToE, approach, and workload *have* some ToE parameters, approach parameters, and workload parameters respectively. Additionally, an approach can *have* some approach kind. Finally, each evaluation *has* some date and *performed by* some organisation.

The UML Representation

In this section, we define a profile used by an expert in order to capture performance evaluation results called Generic Evaluation Model (GEM). This profile is depicted in Figure 4.8.

The GEM profile consists of nine stereotypes and nine relations which are direct mappings of the elements of the core evaluation ontology. The preserved classes are *gemToE*, *gemToE_RBB*, *gemToE_Function*, *gemEvaluation*, *gemApproach*, *gemWorkload*, *gemMetrics*, *gemPlatform*, and *gemRequiredComponent*. The preserved relations are *uses*, *evaluatesUnder*, *obtains*, and *evaluates* which relates gemEvaluation with gemApproach, gemWorkload, gemMetrics, and gemToE respectively. Other directly mapped relations are *measuredOn* between the gemMetrics and gemPlatform stereotypes, *contains* between gemPlatform and gemRequiredComponent, *executesOn* between gemToE and gemPlatform, and *provides* between gem-

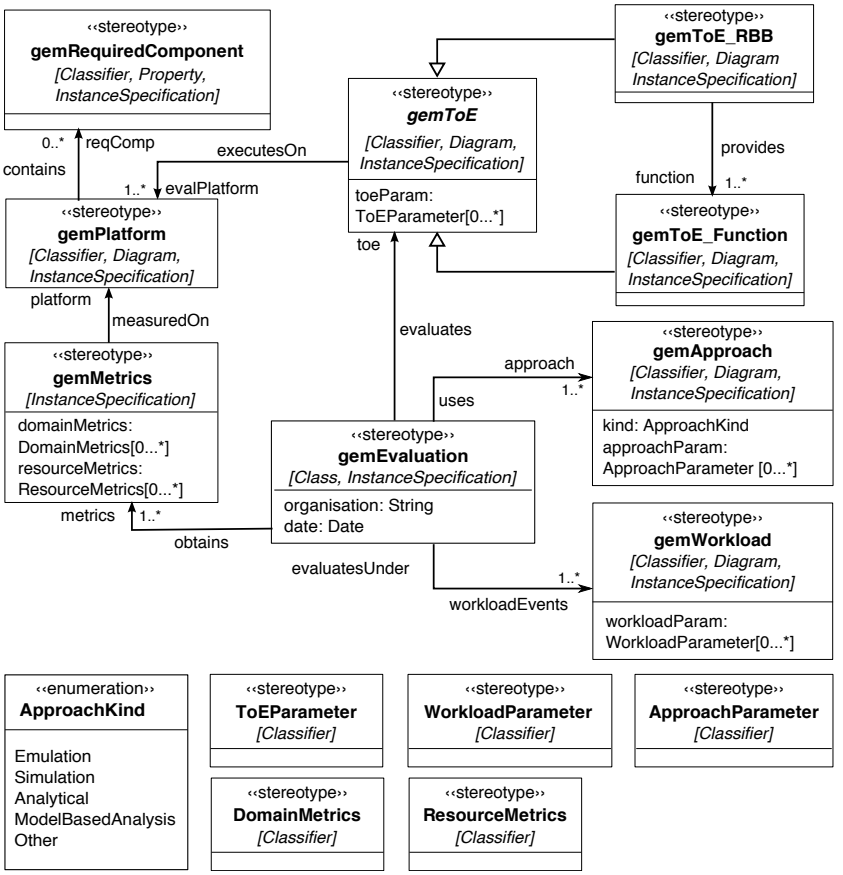


Figure 4.8: Generic evaluation model UML profile

ToE_RBB and gemToE_Function. The is-a relations from *gemToE_RBB* and *gemToE_Function* to *gemToE* in the core evaluation ontology are modelled as generalisations in GEM.

Other elements of the evaluation ontology are specified in a different way. Individuals of the approach kind class are represented as an *ApproachKind* enumeration (not shown). The relation between the ToE and ToE parameter concept in our ontology are represented by the tag definition *toeParam* in the stereotype gemToE. Likewise, the *approachParam* tag definition in gemApproach replaces the *has* relations between the approach and approach parameter concepts of the core evaluation ontology. The same logic applies for the workloadParam tag definition. Finally, the *has* relation between the metrics concepts are represented as corresponding tag definitions of the gemMetrics stereotype, namely *domainMetrics* and *resourceMetrics*.

In the following, we explain what UML models can be annotated with the stereotypes from GEM, and what MARTE packages are used. We conclude

this section with an example illustrating a possible use of the GEM profile.

gemToE, *gemApproach*, and *gemWorkload* can be used to annotate either a complex or a very simple UML model of a corresponding constituent. The level of detail does not play a significant role for the GEM profile. Nevertheless, the richer these models are the more informed decisions can be made by an embedded system engineer when selecting RBBs.

In order to model an evaluation platform, i.e. the gemPlatform stereotype, we use a UML class model annotated with stereotypes from the HRM MARTE package. An execution platform can describe resources that take a variety of forms, e.g. hardware, software, or logical resources. In this study, we consider only hardware components. However, the general concept is scalable to include other forms of resources for analysis.

We employ the non-functional properties (NFPs) modelling package of MARTE to specify parameters and metrics tags defined in GEM. These are such classes as *ToEParameter*, *ApproachParameter*, *WorkloadParameter*, *DomainMetrics*, and *ResourceMetrics*. The NFP package provides a domain model and syntax for specifying different kinds of customer data type. MARTE also provides a library with predefined primitive types and types commonly used in real-time and embedded system domain (see Annex D.1 of the MARTE specifications [41]). In short, this library defines such basic data types as interval type and collection type (MARTE_DataTypes package in MARTE_Library) and such NFP types as frequency, data size, and power (BasicNFP_Types in MARTE_Library).

The MARTE GQAM profile (outlined in Section 2.2) allows bridging the gap between model-driven engineering and existing formalisms and tools for analysis. Thus, it can significantly help the domain (in our case, security) experts to design their evaluations. In order to demonstrate how our profile can be used to capture performance evaluation results modelled in GQAM, we identify the correspondence between the stereotypes and tags of GEM and GQAM shown in Table 4.1.

Note that our profile is not restricted to capturing results only when GQAM or its refinements are used. For example, results presented by Preissig [80], that are obtained through a traditional approach, can also be described by GEM (as we will show later in this section). However, GQAM can facilitate this task, since the mapping identified in Table 4.1 can be automated as a transformation directly feeding relevant data into GEM.

GEM is a general UML profile that does not target any concrete extra-functional domain. The same statement is applicable for the core evaluation ontology. Therefore, a domain expert should refine some of its concepts to tailor it to a certain domain. In particular, an expert needs to extend the *ToEParameter*, *DomainMetrics*, and *ResourceMetrics* stereotypes. Figure 4.9 shows such a refinement for the security domain as an example where a cipher and cluster-based anomaly detection RBBs are considered. *ToEParam_Anomaly* and *ToEParam_Cipher* refine the ToEParameter concept. According to these refinements, an anomaly detection mechanism can

Table 4.1: Correspondence between the GEM and MARTE GQAM profiles

GEM	**GQAM**
gemToE, gemToE_RBB, gemToE_Function	Not represented.
toeParam	No direct mapping. This tag definition can be mapped to parametrisation (input) variables declared by an expert within the *GaAnalysisContext* stereotype where *sourceKind* is defined as required (req).
gemEvaluation	Not represented.
gemPlatform	The GaResourcePlatform stereotype.
gemRequiredComponent	Any element from a GaResourcePlatform model can be annotated with this stereotype.
gemApproach	Not represented.
kind (ApproachKind)	Not represented. It is defined by an analysis formalism that underlies the GQAM model.
approachParam	Not represented.
gemWorkload	The GaWorkloadBehaviour stereotype
workloadParam	No direct mapping. This tag definition can be mapped to parametrisation (input) variables declared by an expert within the *GaAnalysisContext* stereotype where *sourceKind* is defined as required (req).
gemMetrics	Not represented.
resourceMetrics	No direct mapping. This tag definition can be mapped to parametrisation (output) variables declared by an expert within the *GaAnalysisContext* stereotype where *sourceKind* is set to calculated (calc), estimated (est), or measured (msr). For example, a range of tag definitions of the *GaStep* stereotype (e.g. throughput, response time, utilisation) can serve for the purpose of the resource metrics.
domainMetrics	No direct mapping. This tag definition can be mapped to parametrisation (output) variables declared by an expert within the *GaAnalysisContext* stereotype where *sourceKind* is set to calculated (calc), estimated (est), or measured (msr).

be characterised by the number of clusters (*numberOfClusters*) and cluster centroid distance threshold (*clusterDistanceThreshold*) [81], while a cipher building block can be characterised by its key size (*keySize*), cipher mode (*cipherMode*), and cipher type (*cipherType*). Quality of service metrics for an anomaly detection are detection rate and false positive rate (see *DM_Anomaly*) and for a cipher they are resistance to attacker's capabilities in terms of its skill, motivation, and duration of the attack(see *DM_Cipher*), and etc. Finally, a pair of resource metrics are considered for these two security RBBs, namely bandwidth and energy (see *RM_Anomaly*) for an anomaly detector and the used memory and data rate (see *RM_Cipher*) for a cipher.

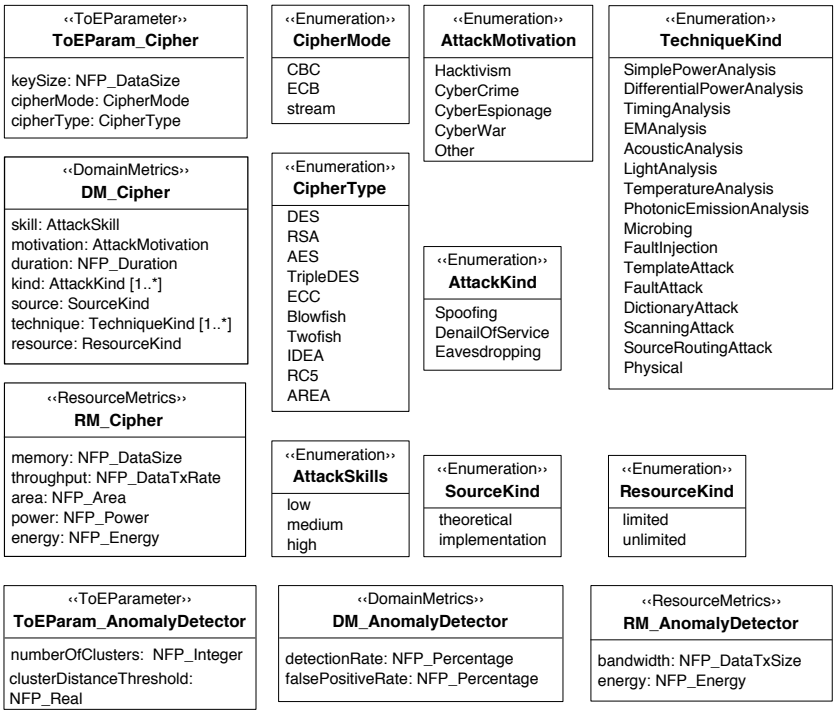


Figure 4.9: The refinement of the GEM profile for the security domain

A description of actual performance evaluation results captured when the GEM profile is used is called Performance Evaluation Record (PER). Figure 4.10 depicts an example of a PER used in development of the smart metering application from Section 2.5. The *DES_Evaluation* class annotated as "gemEvaluation" shows that an evaluated RBB is *DES_Encryption* where *DES_Metrics* is a set of the obtained metrics. The *DES_Encryption* RBB has configuration parameters specified in *DES_Settings* and is executed on the TMS320C6211 chip. Besides, TMS320C6211 is annotated with the *gemRequiredComponent* stereotype and, therefore, will be treated as a plat-

form constraint for *DES_Encryption* in further analysis. For specification of this chip we use two stereotypes from the HRM profile, i.e. *HwProcessor* and *HwComponent*, and one tag, i.e. *r_Conditions*. The two stereotypes intuitively denote that TMS320C6211 is a processing component; the tag associates environmental condition to the component. This information will be further reused for selecting suitable SBBs.

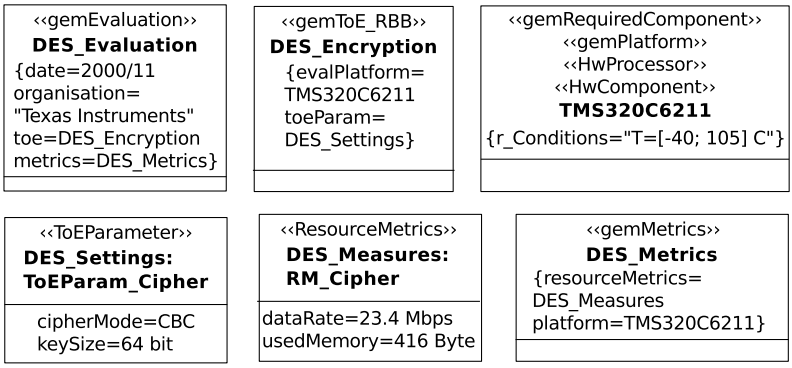


Figure 4.10: Security evaluation record for the DES RBB

Enriched Evaluation Ontology

In the following, we explain the last artefact introduced in Figure 4.6, namely the enriched evaluation ontology.

Each refinement of the GEM profile for an extra-functional domain (e.g. as one depicted in Figure 4.9) is transformed into a separate ontology called *[domain name] evaluation ontology* that imports the core evaluation ontology enriching it with additional concepts from this domain, e.g. refinements of the ToE parameters, domain, and resource metrics concepts. Since each PER (e.g. as one depicted in Figure 4.10) is essentially an instance of the refined GEM profile, we transform it into axioms on individuals and call it *[domain name] evaluation record ontology*.

The transformation is approached similar to one outlined in Section 4.1.1. For example, a class annotated with the ToE stereotype becomes a subclass of a corresponding concept (*ToEParameter*) and an instance becomes an individual of this subclass. A *[domain name] evaluation record ontology* imports a corresponding *[domain name] evaluation ontology*. We refer to a collection of *[domain name] evaluation record ontologies* as an *enriched evaluation ontology*. Figure 4.11 shows the above introduced ontologies and their dependencies. In this figure, *Domain 1* and *Domain n* represent extra-functional domains, e.g. *Domain 1* can be exemplified as the security domain, and *Domain n* as the safety domain.

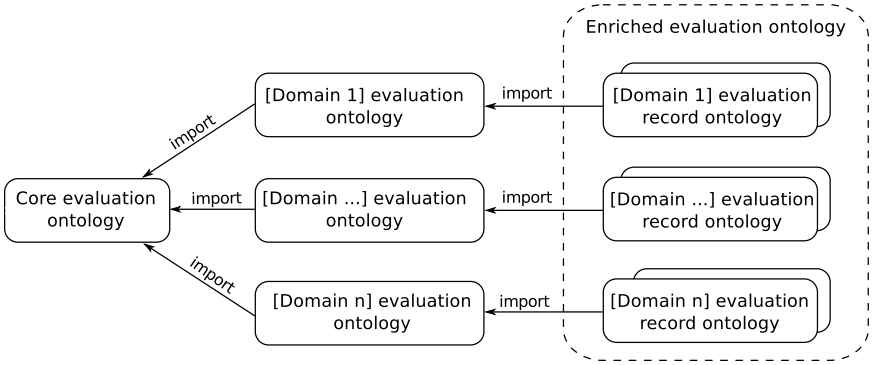


Figure 4.11: Enriched evaluation ontology

4.2 Capturing Security Knowledge

In this section, we explain the proposed process for capturing of the domain-specific security knowledge using the DSSM and PER[2] concepts.

The starting point of the creation of DSSMs is to decide on a domain. The DSML theory inherently leaves the notion of a domain flexible. Hence, it is up to security experts to decide what kind of a domain a DSSM will describe. Typically, we consider application domains (e.g. metering devices, set-top-boxes, banking access terminals), which can be characterised by a different set of assets and a specialised set of security solutions. Domains can be in some relations, e.g. domains can overlap or one domain can be a part of another one. Note that the closer a selected domain is tailored to a type of a system, the more specialised and detailed solutions it contains (i.e. the set of assets and concrete SBBs). For example, both communication and metering DSSMs may be applied for our smart metering devices case study described in Section 2.5, but obviously the communication DSSM will contain such general assets as "message" and "acknowledgement", while the metering DSSM operates with "measurement" as an asset.

The process of DSSM creation is depicted in Figure 4.12. It starts with three activities: *Create a DSSM*, *Create functional models for concrete SBBs*, and *Create PERs for concrete SBBs*. The *Create a DSSM* activity includes definition of assets, abstract SBBs with their goals and strategies, and concrete SBBs omitting the definition of their functional specifications (i.e. the *mFunctional* slot of the *ConcreteSBB* class depicted in Figure 4.3). Thus, the outcome of this activity is a definition of *the DSSM with place holders for concrete SBBs*. Functional models of each concrete SBBs are created during the *Create functional specifications for concrete SBBs* activity. In our case, a security expert creates SPACE models of concrete SBBs. The

[2]In this section, we start again to use the term (concrete) SBB instead of RBB when discussing PER and related concepts shifting the focus back to security.

third activity, i.e. *Create PERs for the concrete SBBs*, is concerned with a description of the performance evaluation results in the form of PERs.

The order of the three activities described above is undefined, since it does not play a significant role for our process. Thus, if SPACE models of the considered concrete SBBs exist (e.g. they are available in the Arctis library of building blocks [43]), the corresponding activity can be omitted. Similarly, the activity *Create PERs for concrete SBBs* can be omitted. The absence of any of the two types of artefacts produced by these activities (e.g. functional models for concrete SBBs or their evaluation records) will simply disable the corresponding type of analysis.

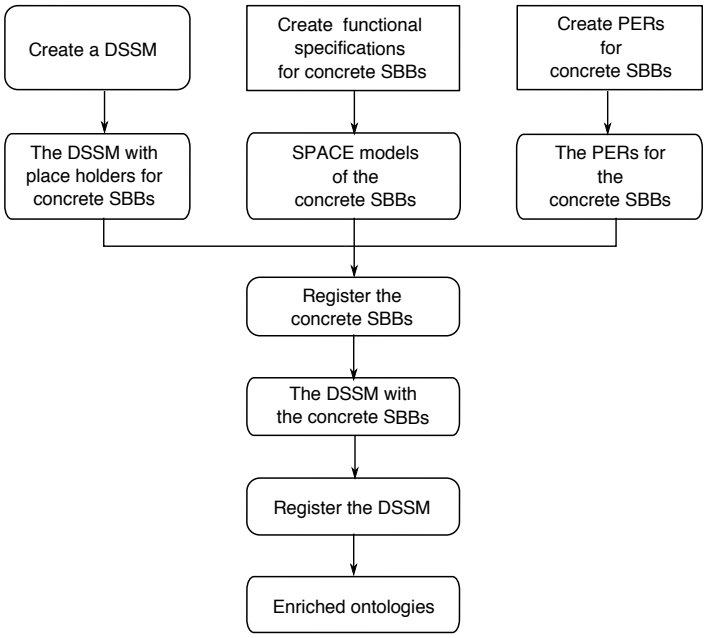


Figure 4.12: The process for creation of DSSMs

The next activity is *Register the concrete SBBs* that allows associating the created functional (SPACE) models and PER models of concrete SBBs with the corresponding elements of the DSSM. In particular, the functional Arctis model is bound to the *mFunctional*. When all concrete SBBs of the created DSSM have been bound with the functional and PER models, the DSSM can be registered.

The main outcome of the *Register the DSSM* activity is two ontologies. The first ontology is a *[domain name] security ontology* derived from the DSSM that is a part of the enriched security ontology as explained in Section 4.1.1. The other ontology is a *[domain name] evaluation record ontology* derived from the PER that is a part of the enriched evaluation ontology as explained in Section 4.1.2. Note that the enriched security and evalua-

tion ontologies are two independent ontologies that can be used separately from each other. In our work, we align them employing the *owl:sameAs* construct (see Section 2.3) as follows: `concreteSBB owl:sameAs ToE`. Figure 4.13 summarises the relation between security and evaluation ontologies, and domain.

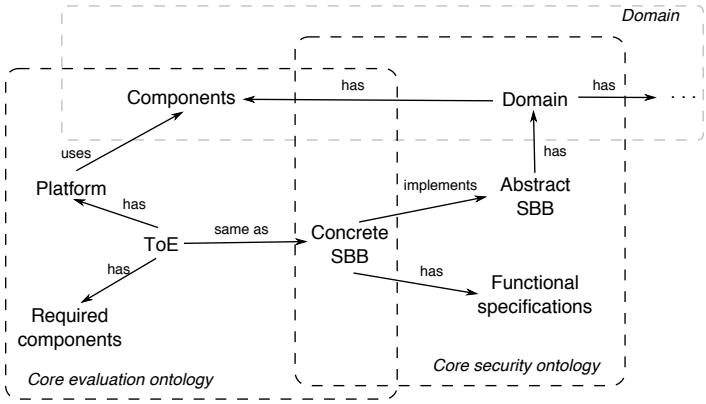


Figure 4.13: Relations between the core security and evaluation ontologies, and domain

It is worth noting that this architecture is modular. For example, the activity *Create PERs for concrete SBBs* can be replaced by any other activity that allows preparing concrete SBBs for a desirable type of analysis. This will require creation of the needed infrastructure (e.g. ontologies and profiles) and adjustment of the alignment.

When updating the enriched ontologies with new knowledge, the important question of maintaining their consistency arises. In particular, an obvious problem when updating the enriched security ontology is its pollution with concrete SBBs that have different names but refer to the same implementation[3]. The unique name assumption of an ontology says that entities with different names refer to different elements of the real world. The OWL language has two constructs to express this assumption, namely *owl:sameAs* or *owl:differentFrom*, that assert that two or more given entities refer to the same or to different elements of the real world respectively. We use the latter construct each time a new concrete SBB is added into the enriched security ontology. However, it may be the case that security experts will populate the enriched security ontology with concrete SBBs that actually refer to the same implementation. This situation can be resolved by the *owl:sameAs* construct that states that two or more individuals refer to the same element of the real world. However, some additional support may be needed to ensure that two (or more) concrete SBBs under different names are equal implementations. We envisage that techniques from the

[3]The trivial case, i.e. two entities with the same names, is not possible.

area of model comparison or models diff (applied to functional and platform models of concrete SBBs) can be employed to address the mentioned issue. For example, Selonen [82] and Bendix and Emanuelsson [83] have a survey about existing model comparison methods for UML models. Besides, a set of tools exist to implement model comparison, e.g. EMF Compare [84]. The exploitation of these techniques goes beyond the scope of this work. We consider it as a further enhancement of our tool support. In the rest of this section, we outline a developed tool that supports the above process of DSSM creation.

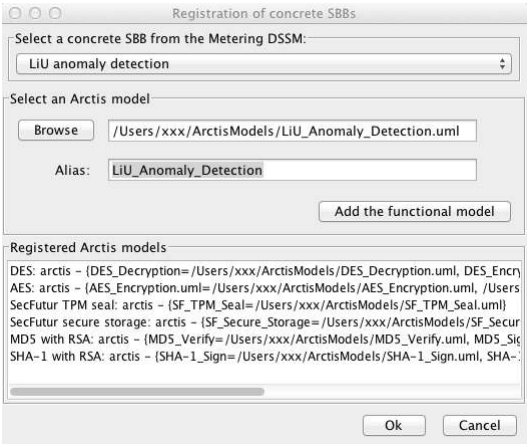


Figure 4.14: Registration of concrete SBBs (the user interface)

As mentioned earlier, we use the MagicDraw tool [13] as an integration environment. To create DSSMs and PERs, we use the standard functionality provided by MagicDraw. At the same time, functional models of concrete SBBs are created in the Arctis tool [42]. In order to bind these two tools supporting the process of the DSSM creation, we have developed a SEED MagicDraw plug-in. In particular, this plug-in assists the creation of DSSMs by supporting the following activities of the process depicted in Figure 4.12:

- Creation of a DSSM: the plug-in prepares an environment for a security expert, i.e. it creates a MagicDraw project and loads the class model depicted in Figure 4.3.
- Registration of concrete SBBs: the plug-in provides an interface (shown in Figure 4.14) for binding functional model and platform model elements of concrete SBBs with corresponding SPACE and MARTE (from PER) models.
- Registration of a DSSM: the plug-in executes transformation of the DSSM and PER to a set of axioms and adds them into the enriched ontologies. Additionally, the plug-in can be used to upload the created DSSM and PER to a library (local or public) for its further use.

This chapter has explained capturing of the domain-specific security knowledge. For this purpose, we have introduced two concepts, namely the DSSM and PER concepts. The process followed by a security expert, that exploits DSSMs and PERs, has been described. In the next chapter, we describe how an embedded system engineer can apply this knowledge to select a suitable set of security solutions to be integrated into a system.

5

Application of the Domain-specific Security Knowledge

This chapter explains the "Development of a security-enhanced embedded system model" activity for the SEED realisation. In particular, we focus on the process of application of the domain-specific security knowledge captured using DSSMs and PERs. This process is depicted in Figure 5.1.

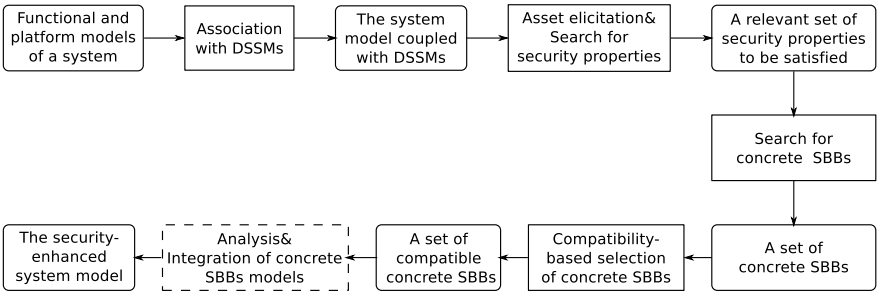


Figure 5.1: Application of the domain-specific security knowledge

The process starts from the functional and platform models of a system. In particular, we use SPACE models (see Section 5.1.1) to create a functional description of a system and UML class models annotated with some MARTE stereotypes (see Section 5.1.2) to create a platform description of a system. Thereafter, a suitable DSSM is selected and its elements are associated with the components of a functional model of a system (see Section 5.2). Note that if a required DSSM is not found, this step should be preceded by

the DSSM creation step explained in the previous chapter. That is, the association step could be postponed until a suitable DSSM is created.

The step *Association with DSSMs* is followed by the step *Asset elicitation&Search for security properties* (see Section 5.3) that results in a list of relevant security properties to be satisfied. Subsequently, concrete SBBs that satisfy the identified security properties are inferred from the enriched security ontology as indicated by the *Search for concrete SBBs* step. Thereafter, a related set of SPACE models are fetched, e.g. from the Arctis libraries. This step is explained in Section 5.4.

Due to the existence of different concrete SBBs, often various ways to secure an embedded system are possible that differ with regards to a range of criteria. For example, an engineer needs to ensure that a system under consideration will still perform the required functionality when security mechanisms are incorporated [47]. Additionally, when dealing with embedded systems, one needs to investigate how the added security mechanisms affect the consumption of crucial resources. The compliance of considered concrete SBBs to some standards can also affect a decision taken by a system engineer. A set of other possible criteria to be considered is proposed by Georg et al. [85]. Thus, to find a suitable solution one needs to carry out analysis of desired criteria and compare different alternatives. In our work, we introduced a so-called model-based compatibility analysis technique applied at the *Compatibility-based selection of concrete SBBs* step. This analysis studies platform-related constraints of a system under development and alternative concrete SBBs.

Subsequently, a set of SPACE blocks that satisfy this criterion are integrated into a system model. At this point, other type of analysis are enabled, e.g. to verify that integration of concrete SBBs provides the required level of protection or that functional properties of a system are not violated by this increment. It is reflected by the *Analysis&Integration of concrete SBBs* step. A dashed line of the box for this step in Figure 5.1 indicates that the elaboration of this step is not a contribution of this work. However, some types of analyses are in a focus of the research group that is developing the Arctis tool.

In the rest of this chapter, we explain each step outlined above using the measurement transfer scenario from the metering infrastructure introduced in Section 2 as a running example. We conclude this chapter with Section 5.6 that gives to a reader a broader view on the process of designing a security-enhanced embedded system. In particular, Section 5.6 extends the proposed process from Figure 5.1 highlighting additional steps and considerations that an embedded system engineer needs account for when integrating security mechanisms into a system.

5.1 System Model

In the following, we demonstrate the languages employed in our work for the functional and platform modelling. Section 5.1.1 illustrates a model of the measurements transfer scenario using the model-based engineering method SPACE. Section 5.1.2 shows an execution platform for a TSMC device defined as a MARTE model.

5.1.1 Modelling a Functional Behaviour of a System

To develop secure networked embedded systems, we employ the model-based engineering method SPACE [2] described in Section 2.2. Recall that applications are composed of *building blocks* that can specify a local behaviour as well as the interaction between several distributed entities. Similar to functional building blocks, security mechanisms can be expressed as self-contained building blocks. These SPACE building blocks that describe functionality of security mechanisms are functional specifications of concrete SBBs introduced in Section 4.1.1. Note that as a result of applying the SPACE method (i.e. building a system as composition of reusable building blocks) the models used in different scenarios can share a lot of commonalities.

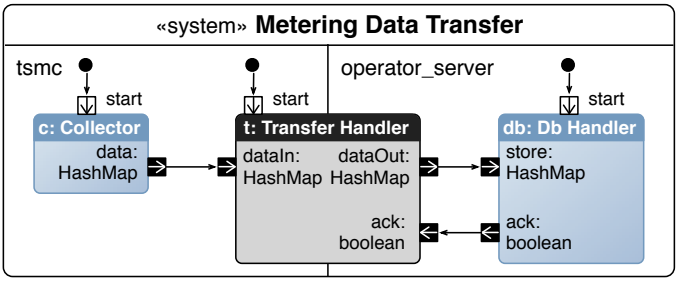


Figure 5.2: Functional model of the measurement transfer scenario

Figure 5.2 depicts the measurement transfer scenario modelled in SPACE. It is a UML activity consisting of two partitions, namely *tsmc* and *operator_server*, that model the respective entities in our scenario. The activity is composed of three building blocks that are connected with some "glue logic" through pins on their frames. The building block *c: Collector* models periodic collection of measurement data from TSMs handled by a TSMC. The block *db : Db Handler* encapsulates the behaviour to store the data in a database of the operator. The block *t: Transfer Handler* manages the communication between the two components that, as will be described later, buffer the data, send it, and resend it in the case of a negative acknowl-

edgement. The block *c* and *db* are local blocks since they specify the local behaviour of an entity. In contrast, the block *t* is a collaborative block as it also describes interaction between two entities. The three blocks (*c*, *db*, and *t*), further, refer to activity diagrams that define their detailed internal behaviour as exemplified for the block *t* in Figure 5.3.

The Petri net-like semantics of the activities models behaviour as control and object flows of tokens between the nodes of an activity via its edges. When a system starts, a token flows from each of the initial nodes (•) following the edges of the activity. In the application in Figure 5.2, all three inner blocks are started in the initial node. Then, periodically the collector block emits a token containing an object of type *HashMap* through its pin *data*. This object maps TSM identifiers to measurement values at a particular time. As depicted by the outgoing edge from pin *data* of block *c*, the object is forwarded to block *t* and further to block *r: Reactive Buffer* via its pin *add* (see Figure 5.3). This buffering block, which is taken from one of the Arctis libraries, is used to buffer measurement data that may arrive when other data is being sent but not yet acknowledged. If data is received when the buffer is empty, it is emitted immediately; otherwise, it is buffered. The pin *next* is used to get subsequent data. Following the outgoing edge of the pin *out* of the block *r*, a copy of measurement data is stored temporarily in variable *temp* by the operation *set temp*. Thereafter, the token flows through the merge node (◇) and data is sent to the other entity as illustrated by the edge crossing the partition border. In the receiver partition, the data is forwarded out of the block which, according to Figure 5.2, is stored in a database by the block *db*.

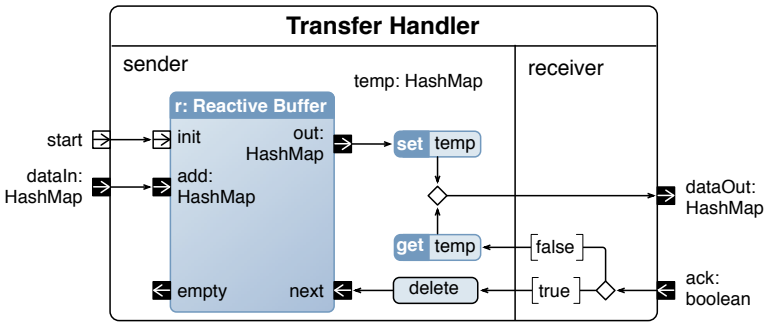


Figure 5.3: Detailed behaviour of the transfer handler block

The block *db: Db Handler* in Figure 5.2 will emit a token via the pin *ack* containing either a positive or a negative acknowledgement. A positive acknowledge corresponds to a successful transfer of measurement data, while a negative acknowledgement is issued in the case the received measurements have not passed the validation test executed by the *db* block. Thus, the token

emitted via the pin *ack* flows further inside the block t and, as depicted in Figure 5.3, reaches the decision node ($\diamond$). A positive acknowledgement leads the token flows through the outgoing edge labelled with *true*. Thereafter, the operation *delete*, that removes the data stored in the variable *temp*, is called and subsequent data, if any, is retrieved from the buffer r. A negative acknowledgement moves the token through the edge labelled with *false*. In this case, the previously sent data is retrieved from the variable *temp* and sent again.

5.1.2 Modelling an Execution Platform

A platform model of a TSMC device for our scenario is depicted in Figure 5.4. We use UML class models annotated with stereotypes of the Hardware Resource Modelling (HRM) package of the MARTE profile (see Section 2.2). The modelling is done in the MagicDraw tool [13].

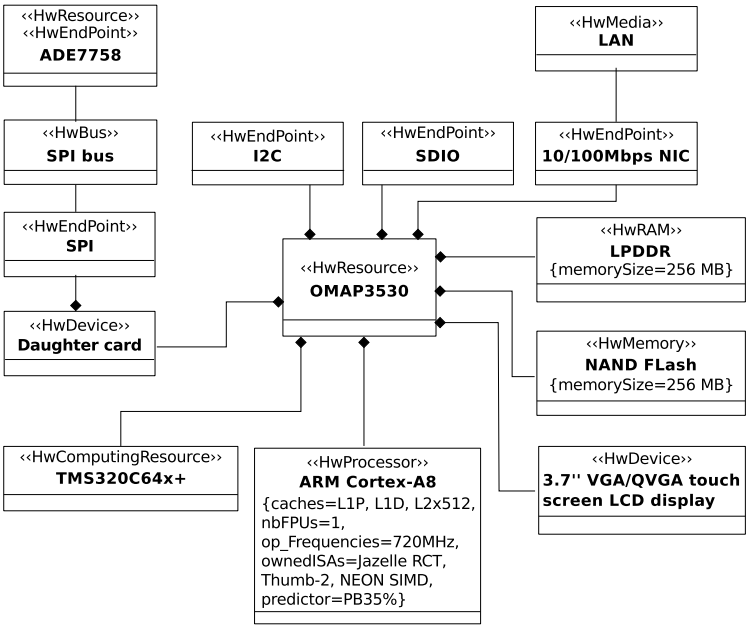


Figure 5.4: Platform model for a TSMC device

The main component of a TSMC platform is an OMPA3530 board [86]. This board includes computing elements (TMS320C64x+ DSP and ARM Cortex-A8), storage elements (NAND Flash and LPDDR), communication interfaces (I2C, SDIO, and 10/100Mbps NIC), a daughter card, and a LCD display. The daughter card is connected to the ADE7758 sensor via a Serial Peripheral Interface (SPI) bus. Finally, 10/100Mbps NIC is used to connect a TSMC to a communication channel (LAN). In the following, we briefly

explain stereotypes from the MARTE HRM package used for modelling the TSMC platform.

The *HwResource* is the most general term of the HRM that represents any hardware unit. The *HwComputingResource* denotes a computation unit, where the *HwProcessor* is its refinement that represents a processor or microcontroller unit. Similar, the *HwMemory* is an abstract memory unit that denotes a given amount of memory, where the *HwRAM* refines it to represent a unit of the random access memory. The *HwMedia* is a central term that represents a communication resource used to transfer data over some channel. In our example, we use the *HwBus* stereotype that models a wired channel. The *HwEndPoint* indicates that annotated components are connection points. Finally, the *HwDevice* stereotype refers to an entity that interfaces with an external environment.

5.2 Association with DSSMs

The goal of the step *Association with DSSMs* in the proposed process is twofold: (1) a DSSM that is relevant for a system under development is identified and selected; (2) bounds of a system (its functional model), where the knowledge captured by a selected DSSM should be applied, are established. Figure 5.5 depicts an interface of the SEED MagicDraw plug-in developed to support this step. We exemplify the *Association with DSSMs* step with our use case of the measurements transfer scenario.

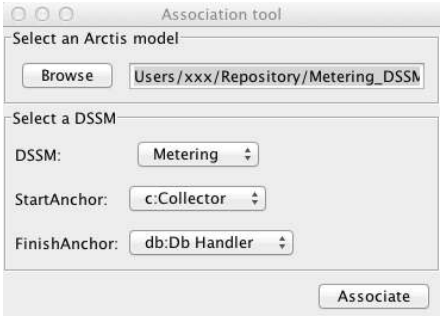


Figure 5.5: Association of the selected DSSM with the system elements (the user interface)

Since the scenario is the smart metering domain, an embedded system engineer selects the metering DSSM from the library of DSSMs (i.e. step (1)). Hence, the association is based on a matching of the system and security domains. Thereafter, those parts of a system containing data to be protected are identified (i.e. step (2)). In this thesis, we discuss the protection of the metering data that, in the functional system model from Figure 5.2, flows from the block *c: Collector* in the TSMC to the *db: Db Handler* in the op-

erator server. Thus, these two blocks are the starting respective end points of the object flow to be protected. Therefore, the identifiers of these two blocks are assigned to the fields *StartAnchor* and *FinishAnchor* respectively as it is shown in Figure 5.5.

5.3 Asset Elicitation and Search for Security Properties

In this section, we present the asset elicitation technique along with the search for security properties method. The developed asset elicitation technique consists of two steps. The first step inspects a functional model identifying present assets. This step uses a set of rules elaborated for traversing a functional model and the information about assets relevant for a certain domain obtained from an associated DSSM. Thereafter, we retrieve from the enriched security ontology a set of security properties associated with the identified assets through the DSSM. Further, we proceed with the second step of the asset elicitation technique. This step utilises a platform model of a system and information about their potential threats to identify vulnerable assets. The output, when the asset elicitation technique is applied, is a set of assets and security properties that associate security goals and defence strategies with the identified set of assets.

In Section 5.3.1 we define a set of rules that allow identifying assets to be protected within a functional system model. Section 5.3.2 shows a method to retrieve a set of security properties relevant for the identified assets. Then, we explain the developed approach to refine a set of identified assets and corresponding security properties utilising the platform description information. This method is presented in Section 5.3.3.

5.3.1 Asset Elicitation on a Functional Model

The first step of the proposed asset elicitation technique is to identify an initial set of security assets utilising a functional model of a system. This identification is implemented as rule-based classification of assets. Thus, the rule-based classification is a method that allows identifying assets within a functional model of a system and their matching to the classes defined in the core security ontology introduced in Section 4.1.1. In particular, the considered classes are “data stationary” and “data in transit”.

The rule-based classification is realised as application of the rules **R1** – **R7** to collaborative-based SPACE models. These rules are presented in Figure 5.6. We have implemented this functionality in a tool called *asset analyser*. Afterwards, an engineer complements this classification according to an associated DSSM. However, it is worth noting that the latter task can potentially be automated given a system modelling language closely tailored to a domain (i.e. a domain-specific language). We now proceed to explain

our rules (see Figure 5.6) and their application logic (see Figure 5.8).

According to the SPACE semantics [44], an activity is a directed graph g with a set of activity nodes V and connecting edges E, i.e. $g = (V, E)$. Figure 5.6 presents the seven identification rules **R1** to **R7**. In the rules, we use the following functions:

- Two functions mapping an activity node and edge to their particular types, i.e. $kind_V : V \to K_V$ and $kind_E : E \to K_E$, where $K_V = \{operation, merge, join, fork, decision, local, collaboration, other\}$ and $K_E = \{object, control\}$.
- The set ON of all object nodes of a given activity, i.e. the data stored in the system and transported within the data flow tokens.
- Two functions mapping a given node to the set of its incoming and outgoing edges, i.e. $in_E : V \to 2^E$ and $out_E : V \to 2^E$.
- Two functions returning an object flowing to (reps. from) a given node through an edge, i.e. $in_O : E \times V \to ON$ and $out_O : V \times E \to ON$.
- A function mapping a node to a set of partitions to which it belongs, i.e. $part : V \to 2^P$, where P is a set of all partitions of a given activity diagram.
- Two functions that return the source and target nodes of a given edge, i.e. $source : E \to V$ and $target : E \to V$ respectively.
- A function mapping a merge node and the set of its incoming object edges to its outgoing object, i.e. $fMerge : V \times 2^E \to ON$. Likewise, we define function *fJoin* for a join node. In SPACE only one outgoing edge is allowed for merge and join nodes [45].
- A function mapping a fork node, its incoming object edge, and one of its outgoing edges to an object flowing through this outgoing edge, i.e. $fFork : V \times 2^E \times E \to ON$. Likewise, we define function *fDecision* for a decision node. For the sake of generality, we allow that the second argument of *fFork* and *fDecision* is a set of edges, but in SPACE fork and decision nodes can have only a single incoming edge [45].
- A function mapping a given asset to a class from the core security ontology, i.e. $class : A \to K_A$, where $K_A = \{transit, stationary\}$ and A is the set of assets constructed from elements of the set ON. Each asset is uniquely identified as a tuple $\langle ON, V, E \rangle$.
- A function that takes a certain activity (i.e. a graph) $g \in G$ and returns a set of assets elicited within this activity g, i.e. $fa : G \to A$, where G is a set of activities.

R1:
$$\frac{\begin{array}{l} q \in V, e \in in_E(q), kind_E(e) = object, \\ kind_V(q) \in \{operation, local, collaboration\} \end{array}}{\begin{array}{l} \exists asset : asset = \langle in_O(e,q), q, e\rangle, \\ class(asset) = stationary, fa(g) = fa(g) \cup asset \end{array}}$$

R2:
$$\frac{\begin{array}{l} q \in V, e \in out_E(q), kind_E(e) = object, \\ kind_V(q) \in \{operation, local, collaboration\} \end{array}}{\begin{array}{l} \exists asset : asset = \langle out_O(q,e), q, e\rangle, \\ class(asset) = stationary, fa(g) = fa(g) \cup asset \end{array}}$$

R3:
$$\frac{\begin{array}{l} e \in E, kind_E(e) = object, \\ part(source(e)) \cap part(target(e)) = \emptyset,\ q \in V, \\ kind_V(q) \in \{operation, local, collaboration\},\ e \in out_E(q) \end{array}}{\begin{array}{l} \exists asset : asset = \langle out_O(q,e), q, e\rangle, \\ class(asset) = transit, fa(g) = fa(g) \cup asset \end{array}}$$

R4:
$$\frac{\begin{array}{l} e \in E, kind_E(e) = object, \\ part(source(e)) \cap part(target(e)) = \emptyset, m \in V, \\ kind_V(m) = merge,\ e \in out_E(m),\ V' \subseteq V, q \in V', \\ in_E(m) \cap out_E(q) \neq \emptyset,\ kind_V(q) \in K_V \setminus \{other\} \end{array}}{\begin{array}{l} \exists asset : asset = \langle fMerge(m, in_E(m)), m, e\rangle, \\ class(asset) = transit, fa(g) = fa(g) \cup asset \end{array}}$$

R5:
$$\frac{\begin{array}{l} e \in E, kind_E(e) = object, \\ part(source(e)) \cap part(target(e)) = \emptyset, d \in V, \\ kind_V(d) = decision,\ e \in out_E(d) \end{array}}{\begin{array}{l} \exists asset : asset = \langle fDecision(d, in_E(d), e), d, e)\rangle, \\ class(asset) = transit, fa(g) = fa(g) \cup asset \end{array}}$$

R6:
$$\frac{\begin{array}{l} e \in E, kind_E(e) = object, \\ part(source(e)) \cap part(target(e)) = \emptyset, j \in V, \\ kind_V(j) = join,\ e \in out_E(j),\ V' \subseteq V, q \in V', \\ in_E(j) \cap out_E(q) \neq \emptyset,\ kind_V(q) \in K_V \setminus \{other\} \end{array}}{\begin{array}{l} \exists assetasset = \langle fJoin(j, in_E(j)), j, e\rangle, \\ class(asset) = transit, fa(g) = fa(g) \cup asset \end{array}}$$

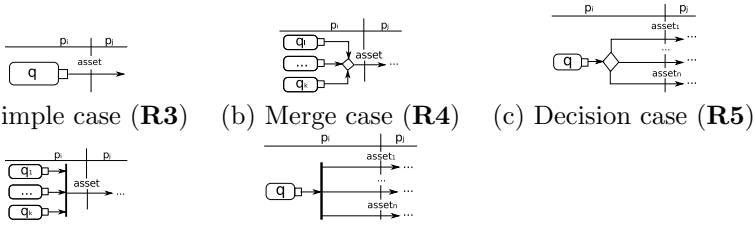
R7:
$$\frac{\begin{array}{l} e \in E, kind_E(e) = object, \\ part(source(e)) \cap part(target(e)) = \emptyset, f \in V, \\ kind_V(f) = fork,\ e \in out_E(f) \end{array}}{\begin{array}{l} \exists asset : asset = \langle fFork(f, in_E(f), e), f, e\rangle, \\ class(asset) = transit, fa(g) = fa(g) \cup asset \end{array}}$$

Figure 5.6: Rules for asset identification

The asset elicitation rules defined in Figure 5.6 form a set of assets for an activity g that is accessed through $fa(g)$. Thus, each time a rule

identifies an asset, i.e. *asset*, this asset is added into a set of elicited assets for g, i.e. $fa(g) = fa(g) \cup asset$. We require that initially (i.e. before the elicitation process starts) the set of assets for an activity g is an empty set, i.e. $fa(g) = \emptyset$. We continue describing in more detail each rule from Figure 5.6 and their application logic.

The rules **R1** and **R2** express that for an operation, local, or collaboration node q a stationary data asset (i.e. *asset*) is observed if this node has an incoming (**R1**) resp. outgoing (**R2**) edge e of the *object* kind. The rules **R3** to **R7** are applied to an object flow crossing a border of two partitions, which corresponds to the data in transit concept. **R3** describes the case that an object leaves an operation node and goes directly to another partition. By the rules **R4** to **R7**, we cover the cases that a flow passes a merge, join, decision, or fork node before crossing a partition border. Figures. 5.7.(a) to 5.7.(e) illustrate the cases of **R3** to **R7** respectively.



(a) Simple case (**R3**) (b) Merge case (**R4**) (c) Decision case (**R5**)

(d) Join case (**R6**) (e) Fork case (**R7**)

Figure 5.7: Illustration of the rules

The application of the rules **R1** – **R7** to a functional system model is outlined by the `traverseBlocks` and `traverseEdges` functions depicted in Figure 5.8. Recall that each activity (i.e. graph) g is represented by a pair of nodes and edges (V, E). First, the function `traverseBlocks` traverses all nodes V and applies the rules **R1** and **R2**. For example, an application of this function to the model in Figure 5.2 will identify six data stationary assets, which are *data*, *dataIn*, *dataOut*, *store*, and two *ack*s.

Thereafter, if a considered node is a local block, the `traverseBlocks` function is recursively applied to its internal behaviour. Likewise, a collaborative block invokes both the `traverseBlocks` and `traverseEdges` functions. For example, this is the case for the analysis of the t block in Figure 5.3. Here, the function `traverseEdges` applies the **R4** rule to the merge node before the crossing edge from partition *sender* to *receiver*. For the decision node before the crossing edges in the opposite direction, the rule **R5** is used. As a result, four data in transit assets are elicited: two *ack* assets incoming to the *get* and *delete* nodes; two *temp* assets outgoing from the *get* and *set* nodes.

Table 5.1 summarises the results of applying the rules to our measurement transfer scenario described in Figure 5.2. For those assets that have

```
function traverseBlocks (Activity g)
    for all v in V do
        R1 − R2
        if kind_V(v) = local then
            traverseBlocks(v)
        end if
        if kind_V(v) = collaboration then
            traverseEdges(v),
            traverseBlocks(v)
        end if
    end for
end traverseBlocks

function traverseEdges (Activity g)
    for all e in E do
        R3 − R7
    end for
end traverseEdges
```

Figure 5.8: Functions to traverse a functional system model

duplicating names (i.e. *ack* and *temp*), we have added their location information in brackets.

Table 5.1: Results of eliciting assets from the functional model

Asset	**DSSM classification**
data, *dataIn*, *dataOut*, *store*, *out*, *add*, two *temp* (to and from the *set* operation), *temp* (from the *get* operation),	StoredMeasurement (Data stationary)
ack (from the *db* block), *ack* (to the *t* blocks)	Not an asset (Data stationary)
temp (from the merge node)	CollectorToServerMsr (Data in transit)
ack (that goes from the decision node to the operations *get*), *ack* (that goes from the decision node to the operations *delete*)	Not an asset (Data in transit)

Figure 5.9 demonstrates the interface of the developed asset analyser tool when it is applied to the scenario of the MixedMode use case from the Figure 5.2. The main output of this tool is a table where each row represents an identified asset. First four columns contain information elicited by the algorithm (the functions and rules) presented in Figure 5.7, namely the name of an asset, its source (the *NodeId* | *EdgeId* column), and its class according to the core security ontology (the *Ontology classification*

column). Classes in the column *DSSM classification* are assigned by an engineer among possible alternative classes captured in an associated DSSM. That is the metering DSSM for this example where the alternative classes are *StoredMeasurement* for data stationary assets, and *CollectorToServerMsr* and *SensorToMeterMsr* for data in transit assets (see Figure 4.4).

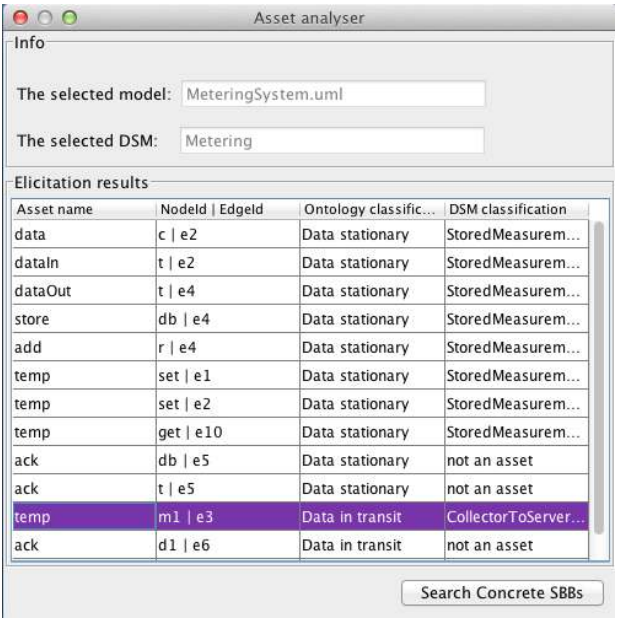


Figure 5.9: Asset analyser (the user interface)

5.3.2 Search for Security Properties

Once the classification of the elicited assets is known a set of corresponding security properties can be retrieved from the enriched security ontology. Recall that the security property concept is defined in the core security ontology as the triple of asset, security goal, and defence strategy. These triples are generated when a DSSM (i.e. an object diagram) is transformed into a set of axioms that are added into the enriched security ontology. Thus, security properties which are available for a given asset can be accessed in the enriched security ontology executing the following query:

```
SecurityProperty and has value [Asset]
```

This query is formulated as an expression written in the Manchester syntax [87]. The Manchester syntax uses the standard description logic notation to specify restrictions (e.g. $\exists$, $\forall$, $\in$) replacing them with the English

language keywords (e.g. `some`, `only`, `value` respectively). All boolean constructs (i.e. $\sqcap$, $\sqcup$, $\neg$) are also replaced by the English language keywords (i.e. `and`, `or`, `not` respectively). The rest of the words in the query defined above are names of the corresponding concepts and relations from the core security ontology depicted in Figure 4.2. The values in square brackets denote parameters of the query. We employ the HermiT reasoner [88] to execute this query.

For example, for those assets that are classified as *CollectorToServerMsr* and *StoredMeasurement* (see Table 5.1) the following set of security properties is retrieved:

```
[CollectorToServerMsr, Confidentiality, Prevention]
[CollectorToServerMsr, Integrity, Detection]
[CollectorToServerMsr, Authentication, Detection]
[StoredMeasurement, Confidentiality, Prevention]
[StoredMeasurement, Integrity, Prevention]
```

5.3.3 Asset Elicitation Utilising a Platform Model

In this section, we explain the second step of the asset elicitation technique. In particular, we present a method that refines the results of the rule-based classification (introduced in Section 5.3.1) by utilising a platform model of a system.

This method is applied when the initial set of assets and related security properties are identified inspecting a functional model. Three steps that compose this method are shown in Figure 5.10. These steps allow identifying which security properties must be considered as those security properties to be satisfied given a platform model. Thus, this extended method allows focusing on a set of relevant assets refining the results from the previous steps (see Sections 5.3.1 and 5.3.2). We use the platform model selected for a TSMC block presented in Section 5.1.2.

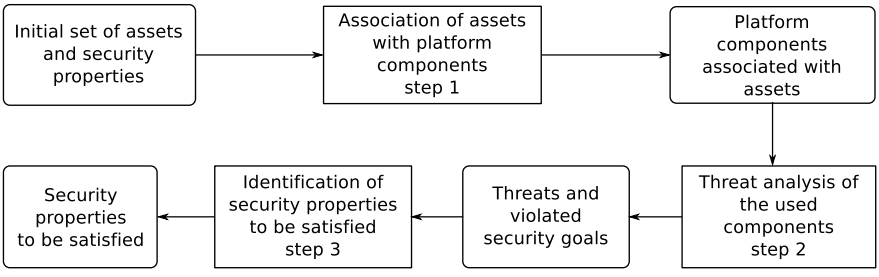


Figure 5.10: Asset elicitation technique utilising a platform model

At the step 1, initially elicited assets are associated with available platform resources, e.g. communication, computing, and storage components.

In general, all platform components that are involved in operations with assets should be mentioned during this association. We provide the following basic guideline:

1. Associate each data stationary asset with a computing unit that operates with this asset (i.e. components annotated with the *HwProcessor* stereotype and its subclasses) and a memory unit that stores it (i.e. components annotated with the *HwMemory* stereotype and its subclasses).

2. Associate each data in transit asset with a communication channel that is used to transmit this asset (i.e. components annotated with the *HwMedia* stereotype and its subclasses) and with two interfaces on the sender and receiver ends (i.e. the *HwEndPoint* stereotype).

3. Associate each data stationary asset with some resource (i.e. components annotated with the *HwResource* or *HwDevice* stereotypes) that contains computing and memory units, which operate with the asset and store it respectively.

4. Associate each data in transit asset with some resource (i.e. components annotated with the *HwResource* or *HwDevice* stereotypes) that contains sender and receiver interfaces and a communication channel, which are involved in transmission of the asset.

Table 5.2 demonstrates association of the assets elicited in Section 5.3.1 with the components of the TSMC platform depicted in Figure 5.4. The *data* asset is associated with the ADE7758 component (the third rule of our guideline). All other data stationary assets (row 2 and 3 in Table 5.2) are associated with the NAND Flash or LPDDR components (i.e. memory units) and the ARM Cortex-A8 component (i.e. computing unit) as it is instructed by the first rule of the guideline. Finally, following the second rule, the data in transit assets (row 4 in Table 5.2) are associated with the 10/100Mbsp NIC components of the TSMC device and of the operator server host (not shown in Figure 5.4) and onto the LAN, which is used as a communication channel.

Alternatively, the required association can be also derived from an allocation model if such is available. For example, MARTE specifies a dedicated package for this purpose that is MARTE::Alloc. It is a powerful package that provides means for capturing spatial and even temporal relations between application and execution platform.

Step 2 from Figure 5.10 involves analysis of existing threats for components of the used execution platform. In general, this task may imply collaboration between a security expert and a system engineer, but the use of threat repositories can facilitate this task. For example, an engineer can query an ontology like one presented by Herzog et al. [72] (potentially extended with other expert knowledge about existing threats). Creation of the

Table 5.2: Association of the assets with the platform components

Asset	Classification	Association
data	StoredMeasurement	ADE7758
dataIn, *dataOut*, *out*, *add*	StoredMeasurement	[NAND Flash, ARM Cortex-A8]
two *temp* (to and from the *set* operation), *temp* (from the *get* operation)	StoredMeasurement	[LPDDR, ARM Cortex-A8]
temp (from the merger node)	CollectorToServerMsr	[OMAP3530: 10/100Mbsp NIC, LAN, DBHost: 10/100Mbsp NIC]

threat ontology goes beyond our scope. Moreover, we find it inappropriate since a set of threat taxonomies, ontologies, repositories, and databases already exist [89, 90]. However, we provide a binding interface that enables integration of needed information into the core security ontology. It is depicted in Figure 5.11. In this figure a threat exploits vulnerabilities that can be present on certain components from a domain. By this means, a threat attacks certain assets and violates certain security goals. In our case, we use knowledge acquired within the SecFutur project combined with the results of the threat analysis for embedded system platforms published by Ravi et al. [91]. Table 5.3 shows the identified threats and potentially violated security goals.

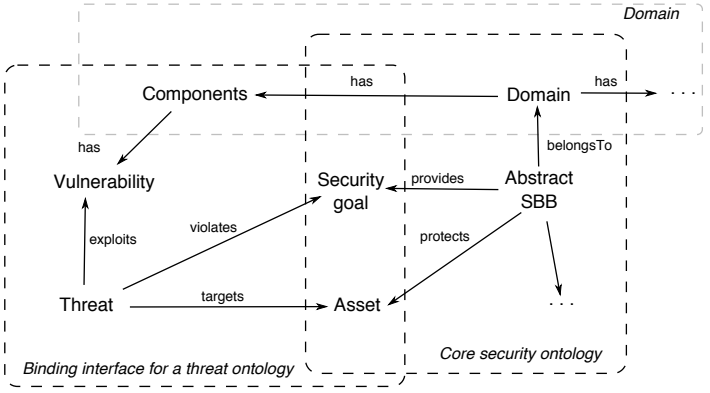


Figure 5.11: Integration of a threat ontology

The last step of the asset elicitation technique (i.e. step 3 in Figure 5.10), automatically identifies a set of security properties to be satisfied. The identification algorithm is implemented as follows. The security goal of each

Table 5.3: Threats and violated security goals

Platform component	Threat	Violated security goal
NAND Flash	injection	integrity
LAN	eavesdropping	confidentiality

earlier retrieved security property (explained in Section 5.3.1) is compared to the security goal violated by the threat (see Table 5.3), which targets a platform component associated with an asset of the considered security property (see Table 5.2). Now, if the security goal of the security property is equal to the security goal violated by the threat, then this security property is added to the set of security properties to be satisfied. In our scenario, we have extracted the following set of security properties:

- SP_1: `[StoredMeasurement, Integrity, Prevention]`
- SP_2: `[CollectorToServerMsr, Confidentiality, Prevention]`

SP_1 is formulated due to the knowledge about the existence of an injection threat that violates integrity of the NAND Flash component (see Table 5.3) used in association of the StoredMeasurement asset (see Table 5.2). Similarly, SP_2 is identified due to the presence of an eavesdropping threat that violates confidentiality of the LAN component (see Table 5.3), which is used in association of the CollectorToServerMsr asset (see Table 5.2).

5.4 Search for Concrete SBBs

At this step, a set of identified security properties to be satisfied are used to find a set of concrete SBBs. They are, for example, the SP_1 and SP_2 security properties for the metering scenario. Concrete SBBs for a particular domain and security properties described within an associated DSSM are retrieved from the enriched security ontology executing the following query[1]:

```
ConcreteSBB and (satisfies value [SecurityProperty])
and implements some (AbstractSBB and belongsTo value [Domain])
```

Execution of the above query for SP_1 and SP_2 retrieves two concrete SBBs for the StoredMeasurement asset, namely *SecFutur secure storage* and *SecFutur TPM seal*[2] and two concrete SBBs for the CollectorToServerMsr asset, namely *AES* and *DES*.

Due to the existence of several alternatives to secure a considered scenario an engineer needs to carry out an additional analysis. This is the case

[1] The query is written in the Manchester syntax [87]

[2] The concrete SBBs that start with the suffix "SecFutur" are developed within the SecFutur [17] European project.

of the measurement transfer scenario above. For example, this analysis may include investigation of the resource overhead introduced by concrete SBBs. Our work contributes to one kind of such analyses proposing a technique that inspects and matches platform constraints of candidate concrete SBBs and constraints of an execution platform model adopted for a design of an embedded system under development. Recall that platform constraints for concrete SBBs are obtained as results of performance evaluation of these security solutions and captured by corresponding PERs that are consequently stored in the enriched evaluation ontology. This step is reflected in Figure 5.1 by the *Compatibility-based selection of concrete SBBs* and presented in the following sections.

Other possible criteria for selection of concrete SBBs could be, for example, their effect on the original functionality of a system and the cost of the concrete SBBs' integration. Formalisation of these needs goes beyond the contributions of this thesis. However, our process accounts this necessity as the *Analysis&Integration of concrete SBBs* step in Figure 5.1.

For the illustration purposes, let us assume that a system engineer decides to use the AES concrete SBB to satisfy SP_2. As a result, the system engineer is directed towards a pair of SPACE blocks, namely *AES Encryption* and *AES Decryption*.

As mentioned in Section 4.1.1 integrating a concrete SBB may create new assets as expressed by the *creates* relation in the core security ontology. Hence, a further search of concrete SBBs, i.e. a recursive application of the above mentioned query, is needed to fulfil the security properties required for these newly created assets. As a result, search of concrete SBBs will continue until all security properties of all created assets are fulfilled. Alternatively, this search will lead to an empty set of SBBs indicating that a vulnerability remains in terms of an unprotected asset. In other words, the search for concrete SBBs results in building alternative sets of concrete SBBs. Each set satisfies a considered security property.

Note that such an approach can lead to a cycle since an ontology reasoner exhaustively searches for any concrete SBB in a DSSM that satisfies the security property. For example, the query can return the same concrete SBB that has invoked it if this concrete SBB satisfies the same security property required by its created asset. To handle such occurrences, we have used an algorithm that detects and resolves such cycle conditions.

This algorithm is based on constructing a directed graph while the search for concrete SBBs goes on:

- Create a node for each found concrete SBB and asset.
- Create a directed edge from a concrete SBB to an asset if the concrete SBB creates the asset.
- Create a directed edge from an asset to a concrete SBB if the concrete SBB protects the asset.

Then, we use a cycle detection algorithm (one based on identification of backward edges during execution the DFS (Depth-first search) algorithm [92]) to detect cycles in the constructed graph. The search continues if there are still alternative paths ignoring (by removing) the detected cycles. Otherwise, the engineer is notified of the remaining unprotected asset.

A nice property of the previously selected SPACE blocks (i.e. the AES pair) is that they already contain a protection of the keys, i.e. new assets are not created. Thus, we can directly continue with the integration of these blocks. The integration of the AES blocks (encryption and decryption) is easily done by arranging their instances before and after the block *t: Transfer Handler* as shown in Figure 5.12.

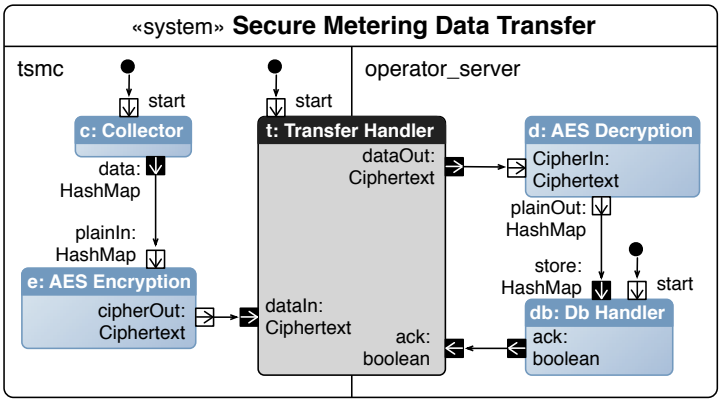


Figure 5.12: Adapted model protecting the transfer of measurement data

The functionality described above is implemented as a tool called *concrete SBB searcher*. The user interface of this tool is depicted in Figure 5.13. In this tool, the list "Security properties" contains a list of relevant security properties for a selected asset (e.g. CollectorToServerMsr). These properties are represented in a form of tuples, i.e. `[asset, security goal, defence strategy]`.

The bottom part of the GUI screenshot in Figure 5.13 contains a tree representation of found sets of concrete SBBs. The root item denotes the selected for the search security property. For this example, we use the earlier retrieved security property SP_2 that is `[CollectorToServerMsr, Confidentiality, Prevention]`. Recall that other security properties have been eliminated when the platform model is added into the analysis (as explained in Section 5.3.3). The root item expands to several items that have the following format:

[asset] requires *[security goal]*: **[concrete SBB]**

In this string, *[asset]* and *[security goal]* are elements from the selected for the search security property, i.e. `CollectorToServerMsr` and `Confidentiality` respectively from SP_2. Then, **[concrete SBB]** denotes a retrieved concrete SBB that satisfies this security property. In our example, they are OpenSSL, DES, and AES implementations. Note that the DES SBB has a special icon, i.e. "CA", that stands for Created Assets. This icon denotes that the DES SBB, in turn, creates some assets to be protected (a key in this example). An engineer needs to expand this item to see the created assets, their required security goals, and available concrete SBBs. Thus, a system engineer can see all possible alternative sets of concrete SBBs that together can be used to protect a considered asset.

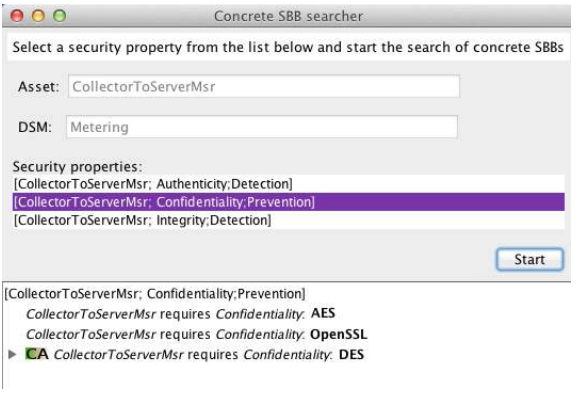


Figure 5.13: Concrete SBB searcher (the user interface)

5.5 Compatibility-based Selection of SBBs

The last step that we describe in this thesis is the *Compatibility-based selection of concrete SBBs* [3]. It is supported by the method for *model-based compatibility analysis* developed in this work. We begin with introducing a reader into the context in Section 5.5.1. In particular, we recall some concepts introduced earlier in this thesis and outline basics of the proposed method. Thereafter, Sections 5.5.2 – 5.5.4 explain the method for model-based compatibility analysis.

[3] In this section, the term SBB can be understood as the term RBB used in Section 4.1.2

5.5.1 Introduction into the Compatibility Analysis

Adding a new feature to a system always comes with resource claims. In Section 4.1.2, we introduced the PER (Performance Evaluation Record) concepts that enables capturing results of performance analysis of SBBs conducted by domain security experts. Among other information, PERs store data about resource footprint and quality of extra-functional properties provided by concrete SBBs. Thus, an embedded system engineer can use this information already at the early design phase increasing the efficiency of an embedded system design process. Hence, the PER concepts contributes to selection of a suitable set of concrete SBBs by enabling the sensitivity and trade-off analyses at early phases.

Another useful and significant input to the design process is knowledge about platform-specific constraints of concrete SBBs. These constraints originate from the fact that SBBs (and just RBBs) for embedded systems are often optimised to exploit a particular feature of hardware components on which they are implemented. Thus, they can provide good quality of extra-functional properties (i.e. level of security) while consuming a small amount of limited resources. In our studies, we argue that these constraints also need to be documented and accounted in order to support integration of concrete SBBs into embedded systems. For example, Preissig [80] reports the results of performance analysis of the Data Encryption Standard (DES) implementation optimised for the memory architecture of the used chip. Similarly, other implementations of DES rely on the presence of a certain instruction set to accelerate permutation operations [6]. Therefore, this information is included as a part of a PER using the *gemRequiredComponent* stereotype from the GEM profile explained in Section 4.1.2.

Consequently, each PER is transformed into an ontology called enriched evaluation ontology that allows storing and structuring this field knowledge. As a result, a variety of information can be retrieved from this ontology defining appropriate queries to the ontology, for example:

- Retrieve values of relevant performance metrics for a certain SBB.
- Retrieve a set of SBBs of a particular domain that satisfy required values with respect to certain performance metrics.
- Retrieve a set of SBBs, for which the platform constraints are compatible with a platform adopted for an embedded system under development.

Queries like the first two can be directly implemented as SPARQL queries [65]. However, the task of compatibility analysis, i.e. the third query in the list above, requires more sophisticated support. Consequently, we develop such a support as a method that implements *model-based compatibility analysis*.

This analysis allows automatically accounting for platform constraints while selecting a set of SBBs to be used for a system design. An extra-functional domain expert (in our study, a security expert) provides these constraints by annotating elements of an evaluation platform (i.e. a MARTE model) with the *gemRequiredComponent* stereotype (see Figure 4.8).

The core of our method is a set of ontologies and SPARQL queries designed to infer whether concrete SBBs and an adopted platform for the embedded system are compatible (i.e. whether the formulated platform constraints for a concrete SBB fits platform declarations of the system being configured). The SEED MagicDraw plug-in implements this functionality to support the use of the compatibility analysis.

In the following, Section 5.5.2 explains the developed hierarchy of ontologies and Section 5.5.3 defines the notion of compatibility. Section 5.5.4 shows results of scalability and performance estimations for our approach.

5.5.2 Ontologies for Compatibility Analysis

Figure 5.14 depicts the developed set of ontologies. These ontologies are organised in three layers: expert, vendor, and engineer layers. The name of each layer denotes its main actor. These ontologies are related to each other through the *import*, *use*, and *refer to* relations.

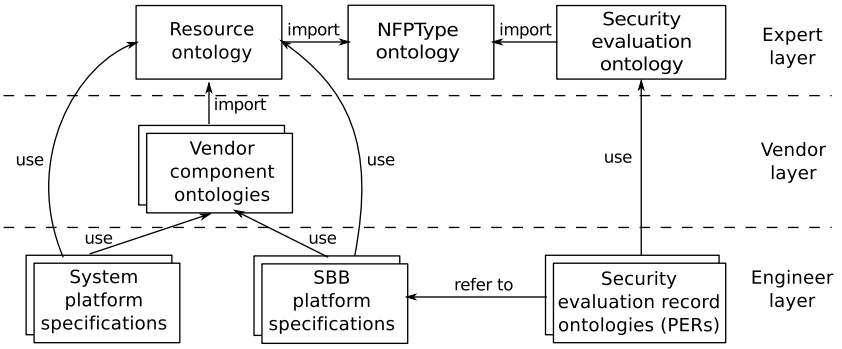


Figure 5.14: Ontologies for compatibility analysis

Expert Layer

Ontologies of the *Expert layer* are created and maintained by experts of the embedded and security (or any other extra-functional properties) domains. It contains three ontologies. The first two ontologies (from the left to the right) are obtained from the transformation of MARTE packages dedicated to platform resource descriptions. The third ontology is the refinement of the core evaluation ontology for the security domain as it is already explained in Section 4.1.2.

Techniques to transform UML class models into ontologies are studied and presented by several researchers, e.g. by Hermida et al. [93] and Xu et al. [94]. Similar to their works, we identify the mapping needed to transform MARTE packages. Basic mapping rules that we apply in our work may be summarised as follows:

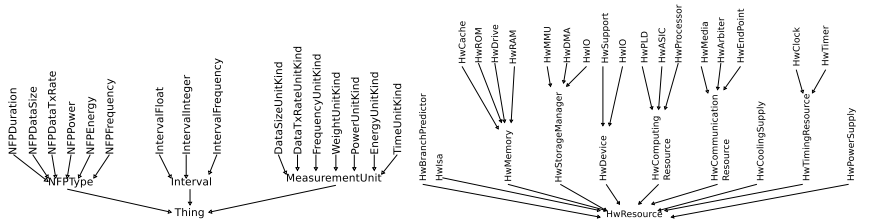
- Each MARTE stereotype is represented as an OWL class.
- Each tag definition is represented as an OWL object or data property, where the domain is the stereotype that owns the tag and the range is the type of the tag. We create an OWL object property if the type of the tag is defined as another stereotype, which can not be replaced with a basic type defined in XSD [95] (e.g. float, integer, or string). Analogously we create an OWL data property if the type of the corresponding tag is a basic XSD type.
- An enumeration is represented as a class with a predefined set of individuals.
- The generalisation relations of the MARTE profile are represented as sub-class relations in the ontology, i.e. the "is-a" relation.
- The composition relations are represented as the part-whole object properties [96], i.e. the "hasPart" relation.
- The named associations are represented as OWL object properties with the corresponding name, e.g. the "hasConnected" relation.

We avoid using the Ontology UML profile [97] that allows designing ontologies as UML models since this requires in-depth understanding of the underlying ontologies from an engineer. Our goal is to exploit advantages of ontology technologies (e.g. querying services), but to allow an engineer to operate only with terms of a considered extra-functional domain.

We proceed to explain the remaining two ontologies that build the expert layer, i.e. *NFPType ontology* and *Resource ontology*. The third, *Security evaluation ontology*, was already discussed in Section 4.1.2.

The *NFPType ontology* of the expert layer contains a set of types and their relations needed to characterise a piece of hardware/software component, e.g. data rate (Mbps, Kbps, etc.) and frequency (Hz, KHz, etc.). This ontology is derived from the MeasurementUnits, MARTE_DataTypes, and Basic_NFP_Types sub-packages of the MARTE_Library package (chapter D.2 of the MARTE profile specification), which enable specification of non-functional properties.

Note that types such as NFP_Real and NFP_Integer are not present in our NFPType ontology since they can be sufficiently modelled as the XSD types [95], i.e. xs:float and xs:integer respectively. Figure 5.15(a) depicts an excerpt of the classification from this ontology.



(a) NFPType ontology (b) Resource ontology

Figure 5.15: Classification levels for the developed ontologies (excerpts)

The *Resource ontology* contains those concepts needed to describe platform components and is derived from the MARTE HRM package described in Chapter 14.2 of the MARTE profile specification [41]. Some classification levels of this ontology are depicted in Figure 5.15(b).

The core concept of our ontology is the HwResource class that denotes a generic hardware entity. The HRM package differentiates two complementary views onto components, namely logical and physical. This structure is flattened in our ontology.

The logical view (Hw_Logical model) consists of five sub-packages. The core elements of these sub-packages are transformed into sub-classes of the HwResource class. These classes are HwComputingResource, HwIsa, and HwBranchPredictor from the Hw_Computing sub-package; HwMemory and HwStorageManager from Hw_Storage; HwCommunicationResource from Hw_Communication; HwTimingResource from Hw_Timing; and HwDevice from Hw_Device.

The physical view (Hw_Physical model) contains two sub-packages, namely Hw_Power and Hw_Layout. The core concepts of Hw_Power are represented as sub-classes of the HwResource class, i.e. HwPowerSupply and HwCoolingSupply. The Hw_Layout model contains only one element, i.e. HwComponent. This stereotype is used to specify physical characteristics (e.g. weight and area) and environmental conditions (e.g. temperature and humidity) for a platform component. We treat it in a special way in our ontology. Thus, we have encapsulated physical characteristics of a component into the "ComponentCharacteristics" OWL class and related it to the HwResource class with the "hasCharacteristics" object property.

The composition relations defined in HRM are transformed into the "hasPart" object properties [96] (and its inverse property "isPartOf"). The named association "connectTo" is modelled as the "hasConnected" object property (and its inverse property "isConnectedTo"). Both these properties have corresponding sub-properties, e.g. "hasCache" is a sub-property of the "hasPart" property.

Vendor Layer

The *Vendor layer* in Figure 5.14 consists of *vendor component ontologies*, where a vendor is a provider of platform components available for construction of execution platforms for embedded systems (e.g. Texas Instruments [98]). Thus, each ontology encapsulates description of platform components. The resource ontology of the expert layer serves as a description language to describe these components.

In order to provide the description of available components, a vendor needs to perform the following steps. First, a vendor uses the MagicDraw tool [13] to create models of platform components. These models are UML class diagrams annotated with stereotypes from the HRM package. Second, a vendor launches the developed SEED MagicDraw plug-in to transform the created MARTE models of the platform components into an ontology. The user interface of this plug-in is depicted in Figure 5.16(a). We have used the Java OWL API [67] and Acceleo [78] tool to implement this transformation. Figure 5.16(a) depicts an interface of the developed plug-in. A vendor may create a new ontology for described components or add these components into an existing ontology.

Figure 5.4 from Section 5.1.2 depicts a valid model of the OMPA3530 board [86] provided by Texas Instruments.

Engineer Layer

The bottom layer in Figure 5.14 is the *Engineer layer*. At this level, system and security engineers use the ontologies created at the higher levels (i.e. the expert and vendor layers) to model the adopted platform for an embedded systems and platforms used for evaluation of concrete SBBs respectively. In particular, an engineer uses vendor ontologies when certain vendor components required for the application are known, whereas the resource ontology is suitable if such a component is not known or not present in vendor ontologies. Security engineers instantiate the security evaluation ontology to capture the results of performance evaluation of a considered SBB. Thus, security engineers create *PERs* where the concept *gemPlatform* refers to the *SBB platform specification* (see Figure 5.14). These PERs are further transformed into corresponding security evaluation record ontologies as explained in Section 4.1.2.

In the smart metering infrastructure from Section 2.5, TSMC devices are built on the OMAP3530 board depicted in Figure 5.4. This component will be already described and stored in the vendor ontology. Therefore, an embedded system engineer needs to load the ontology and use this component as a part of the TSMC model. The developed SEED MagicDraw plug-in supports this functionality.

In the measurement transfer scenario analysed in Section 5.3, the data transmitted between a TSMC and server (i.e. the CollectorToServer asset) must be protected against confidentiality threats as it is indicated by the

security property SP_2. Consequently, two concrete SBBs that satisfy this security property have been found in the Metering DSSM (see Section 5.4). They are the AES (Advanced Encryption Standard) [99] and DES (Data Encryption Standard) [80] implementations provided by the Texas Instruments [98]. For these implementations, the AES SBB requires the use of the C64x+ processor, while the DES SBB requires the use of the TMS320C6211 chip (see Figure 4.10 in Section 4.1.2).

In the next section, we explain how this architecture of different ontologies enables selection of concrete SBBs (or any RBBs) based on the defined compatibility analysis.

5.5.3 Model-based Compatibility Analysis

We differentiate two types of the platform compatibility: *logical* and *environmental*. In the following, we explain each of the mentioned types and exemplify some of them using the system and SBB models introduced in the previous sections.

The notion of *logical compatibility* is based on the pairwise logical compatibility of a SBB and system platform components defined below.

Definition 1 *Two components A and B are* logically compatible *if one of the following holds: (a) A is identical with B; (b) A has B as a part; (c) A is a part of B; (d) A can be connected to B; (e) B can be connected to A; (f) a disjunction of (b)-(e).*

We employ the ontology querying services to automate a check of the above definition. In particular, we use the ASK operator of SPARQL [65] that returns a boolean value indicating whether a path that matches a query pattern exists. For example, the query for case (b) where the relation "hasPart" is examined has the following form:

```
PREFIX hrm:  [the ontology IRI]
ASK {?A hrm:hasPart ?B}
```

Queries for cases (c) – (e) have a similar structure replacing "hasPart" with the "isPartOf", "hasConnected", and "isConnectedTo" object properties respectively. To support the check of case (f), we use a special construct defined by the SPARQL 1.1 syntax, i.e. the so called path properties [100]. It allows examining a path of an arbitrary length. Hence, the query for case (f) replaces the "hasPart" property with a path expression: `(hrm:hasPart | hrm:isPartOf | hrm:connectedTo | hrm:hasConnected)*`. In this expression, the symbol "|" denotes the "*OR*" operator, while the symbol "$*$" means that any number of occurrences is allowed.

In the query mentioned above, the $?A$ and $?B$ symbols denote variables. They are replaced by components of a system platform and components

of a SBB platform (annotated with the "gemRequiredComponent" stereotype) respectively. In our case (see Section 5.5.2), these are OMAP3530 and C64x+ for the AES SBB, and OMAP3530 and TMS320C6211 for the DES SBB. Since TMS320C64x+ has a C64x+ processor as its part, the query returns *true*. In contrast, no path is found between TMS320C6211 and TMS320C64x+ for the DES SBB. Thus, we conclude that the particular implementation of the DES algorithm is not logically compatible with the current design of a system platform that is based on the OMAP3530 board, while AES can be selected as a SBB to provide secure communication for a TSMC device.

The definition of *environmental compatibility* is built upon the *Env_Condition* data type from the HRM package (see Figure 14.72 from the MARTE specifications [41]) which defines five types of environmental conditions: temperature, humidity, vibration, shock, and altitude. Its semantics is a safety condition applied to a component. An environmental condition of each type has a value range and also is applied to a certain state of a component. An engineer needs to annotate the components with the "HwComponent" stereotype and define the "r_Conditions" tag to assign environmental conditions to a component. We use the following terms and functions to define *environmental compatibility*:

- K, U, and S are sets of the environmental condition types, measurement units, and component states respectively, where $K = \{$*temperature, humidity, vibration, shock, altitude*$\}$, $U = \{^{\circ}C, \%, m/s^2, g, m\}$, and S=$\{$*operating, storage, other, undef*$\}$.

- A set ENV_COND defined as $I \times U \times K \times S$ that describes each environmental condition as a tuple of a value interval (a set I), a unit (from the set U), a type (from the set K), and a state (from the set S).

- A function defining environmental conditions of a component, i.e. $env\_cond : COMP \rightarrow 2^{ENV\_COND}$, where $COMP$ is a set of components.

- Projection functions extracting from an environmental condition the corresponding type, i.e. $kind : ENV\_COND \rightarrow K$, unit, i.e. $unit : ENV\_COND \rightarrow U$, state, i.e. $state : ENV\_COND \rightarrow S$, and value interval, i.e. $range : ENV\_COND \rightarrow I$.

Given these terms and functions we introduce two other functions to define the notion of environmental compatibility.

Definition 2 *Given that environmental compatibility is a function* $env\_comp : COMP \times COMP \rightarrow \{true, false\}$ *then a component A is* environmentally compatible *with a component B if* $env\_comp(A, B)$ *is evaluated to* true *as defined below:*

$env\_comp(A, B) \triangleq true$ *if* $\forall k \in K, \forall s \in S : a \in env\_cond(A),\ b \in env\_cond(B),\ kind(a) = kind(b) = k,\ state(a) = state(b) = s,\ range(a) \cap range(b) \neq \emptyset$

The intuition is that environmental conditions of a platform component A adopted for an embedded system and a component B required by a SBB are compatible if corresponding interval values of environmental conditions of the same type are overlapping. Note that if some conditions are not specified for a component we assume that an interval value for such a condition spans over its whole admitted range.

In addition, we define another function that specifies the environmental conditions under which a pair of components can not operate (although each could operate individually under respective conditions). We refer to such environmental conditions as environmental constraints.

Definition 3 *Given two components A and B,* environmental constraints *for these two components are returned by the function* $env\_constr : COMP \times COMP \rightarrow 2^{ENV\_COND \times ENV\_COND}$ *defined as follows:*

$env\_constr(A, B) \triangleq \{\{\langle i_1, u, k\rangle, \langle i_2, u, k\rangle\} \mid a \in env\_cond(A), b \in env\_cond(B),\ k \in kind(a) \cap kind(b),\ i_1 = range(a) \setminus range(b),\ i_2 = range(b) \setminus range(a),\ u = unit(a)\}$

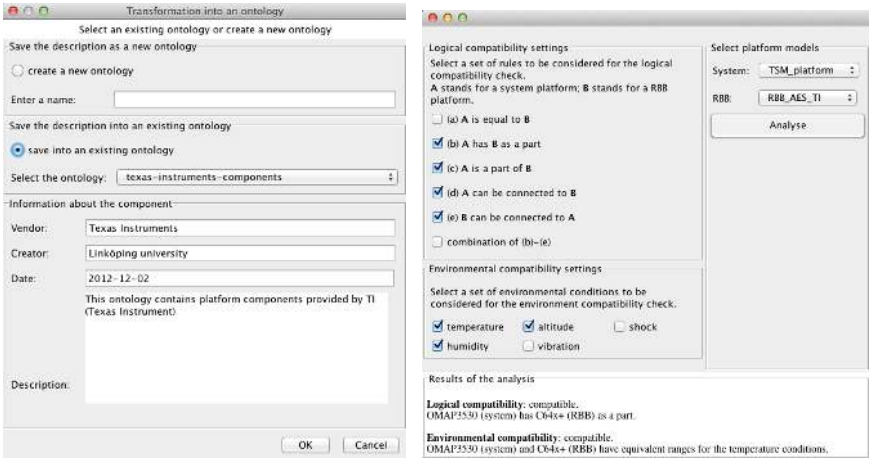
The intuition is that while each component might operate in an interval that is a subset of its operational condition, its composition with another component dictates that one component is prohibited from operating in those environment conditions that are not within the range allowed by another component (and vice verse). In other words, environment conditions of one component are constrained because of presence of another component. Therefore, environment constraints for each component are generated. Consequently, the presence of these constraints for components guides a system engineer to implement a mechanism (e.g. cooling system) that ensures that the corresponding components sustain the environment constraints generated for another component (e.g. to keep within a given temperature range). The SEED MagicDraw plug-in generates these environmental constraints automatically from given environmental conditions attached to individual components.

In our scenario, the temperature conditions for OMAP3530 and TMS32-0C64x+ (the AES SBB) have the same ranges of $[0; 90]^{\circ}C$. Therefore, the system and SBB are environmentally compatible without any additional constraints.

The above definitions for computing the compatibility relation and their pairwise imposed constraints allow us to reason about environmental conditions of assemblies based on constraints for its constituent components.

Figure 5.16(b) depicts the user interface of the SEED MagicDraw plug-in supporting compatibility analysis. It allows selecting a type of the desired

compatibility analysis (logical and environmental) and its settings. The bottom part of this tool shows the results of the analysis, namely whether the system and SBB are compatible according to the selected criteria. Additionally, it shows a generated set of environmental constraints for the environmental compatibility check.



(a) Transformation tool (b) Compatibility analysis tool

Figure 5.16: Model-based compatibility analysis (the user interfaces)

5.5.4 Scalability and Performance

So far, we have used the scenario from the smart metering infrastructure to illustrate the compatibility analysis and knowledge management ideas supported by our methods and tool. This section proceeds to show that this approach is scalable to domains with large data sets. We design experiments to estimate the potential size of resulting vendor ontologies as well as the execution time for the transformation of MARTE models into OWL.

In this study, we focus on microcontrollers (MCUs) provided by some of popular vendors (Renesas, Texas Instruments, Fujitsu, Atmel, and Microchip Technology). We estimate the potential complexity of corresponding MARTE models and the size of corresponding OWL ontologies in terms of the number of generated axioms. Three classes of embedded systems and MCUs commonly used for their design [101] are considered: small scale (8-bit MCUs), medium scale (16-bit MCUs), and sophisticated embedded systems (32-bit and ARM-based MCUs). Thereafter, we study how many models are currently available on the market for each vendor (see Table 5.4). The data has been extracted from the official Internet resources of the vendors mentioned above.

Table 5.4: Scalability and performance estimations

		8-bit	**16-bit**	**32-bit**
1	Renesas	933	2290	1817
2	Texas Instruments	0	406	292
3	Fujitsu	103	207	630
4	Microchip Technology	348	334	79
5	Atmel	238	0	179
6	Total amount of units	1622	3237	2997
7	Av. number of axioms per unit	68	105	133
8	Approx. total number of axioms	110 296	339 885	398 601
9	Av. transformation time (ms)	1455	1627	2497

To estimate the potential number of generated axioms, we select five commonly used MCUs of each class, create their MARTE models, and execute their transformations. This study shows that the simplest 8-bit MCUs creates an average of 68 ($\sigma = 3$) axioms and 16-bit MCUs correspond to 105 ($\sigma = 6$) axioms while the most sophisticated 32-bit MCUs generate 133 ($\sigma = 21$) axioms (see Table 5.4, row 7). The 8th row shows the approximate number of produced axioms when all models are added into ontologies. Finally, we compare these numbers with scalability studies of the OWL APIs and Jena technologies done by Horridge and Bechhofer [68] and Dibowski and Kabitzsch [102]. In particular, Horridge and Bechhofer [68] show that OWL APIs can easily handle ontologies that contain 1 651 533 axioms consuming 831 MB. As a result, we conclude that the used technologies (OWL APIs and MARTE) allow handling ontologies for a significant number of vendors in a potential real world deployment. This capacity allows loading multiple vendors' ontologies to execute compatibility analysis. Additionally, some techniques for swapping ontologies in memory can be implemented to handle even bigger datasets.

Next, we execute 50 runs of the transformation for the same representatives of each MCU class and measure the execution time for each run (see Table 5.4, row 9). In particular, we measure the execution time of the following operations for each run: transformation of an original UML/MARTE model into the OWL API syntax; generation of axioms executing corresponding OWL APIs; and saving the resulting ontology into an owl-file. The hardware used in this study is a system with 2.8 GHz Intel Core i7 and 8 GB of RAM running Mac OS X 10.7. In our case, the transformation time does not vary substantially for small ($1455ms, \sigma = 253$) and medium ($1627ms, \sigma = 292$) MCUs while one-second increase is observed for the sophisticated 32-bit MCUs ($2497, \sigma = 328$). This increase can be explained by naturally larger, in comparison with 8-bit and 16-bit MCUs, complexity of 32-bit MCUs in both number of elements and their attributes.

5.6 Extended Form of the Process

We have presented the process for application of the domain-specific security knowledge that is depicted in Figure 5.1. The process has been intentionally simplified by the author to facilitate explanation of the introduced methods and tools. In this section, we give a broader view to this process.

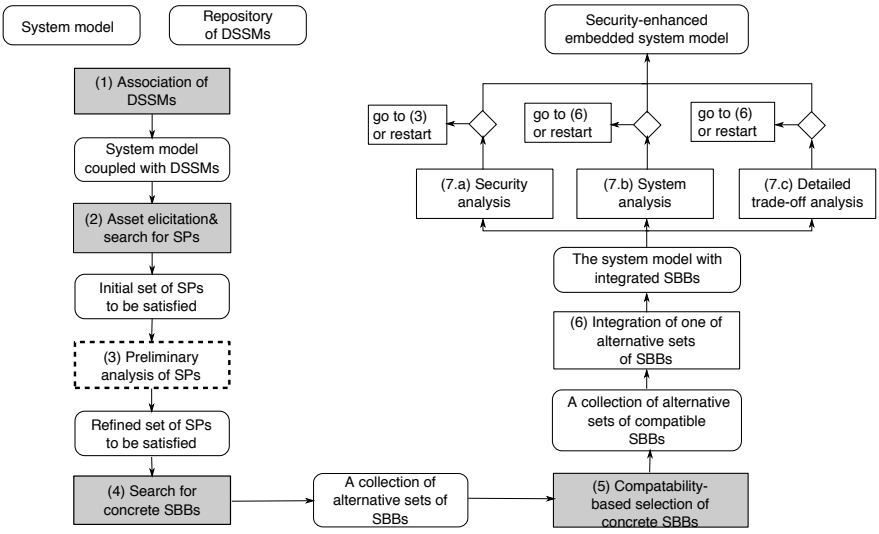


Figure 5.17: Extended form of the proposed process

Figure 5.17 depicts the extended representation of our process. Input artefacts of this process are a system model created by a system engineer and a repository of DSSMs created by domain security experts. A system model includes both functional and execution platform models. The grey boxes indicate the steps that rest on the SEED methods explained earlier in this chapter.

In step 1 a suitable DSSM is selected and boundaries of a system, where knowledge captured by a selected DSSM should be applied, are established (see Section 5.2). The association is based on matching of the application domains. The next step is *Asset elicitation & search for SPs* (Security Properties). Step 2 has two elements: (a) a system model is analysed to identify assets that need security protection; (b) the coupled DSSM is consulted to retrieve a set of relevant security properties and an ontology search provides the known tuples in the domain. These methods are described in Section 5.3.

Next, a preliminary analysis of the proposed security properties is conducted in step 3. This step is necessary to eliminate less relevant security properties (that encompass unprioritised assets). The focus of this thesis that we will expand in the next chapter is activities in this part of the process.

Once the set of security properties to be satisfied is fixed, a system engineer proceeds with searching for suitable security mechanisms as indicated by step 4 – *Search for concrete SBBs*. As we have explained in Section 5.4, this step may lead to proposing a number of alternative ways to secure a system, i.e. a collection of alternative sets of concrete SBBs. The *Compatibility-based selection of concrete SBBs* step allows narrowing the initial collection of concrete SBBs sets. It helps to ensure that the platform-related constraints of the system under development are compatible with the SBBs, especially in terms of resource usage and environmental factors.

Thereafter, one selected set of concrete SBBs is integrated into a system design. At this phase, an embedded system engineer can use tools and techniques developed by the research community to conduct a detailed analysis to study consequences of incorporating security functions into a system. The *Security analysis* step can be used to check whether a set of integrated SBBs provides the required level of security. The *System analysis* step can assist in verifying if security functions do not violate functional requirements of a system, e.g. all deadlines are met. The *Detailed trade-off analysis* step is intended to ensure that other non-functional properties are satisfied in presence of the integrated security protection mechanisms, e.g. available system resources are enough to provide the desired level of quality of service (QoS).

Either of these analyses may disclose discrepancies that may sanction a search for another set of concrete SBBs (i.e. go to step (6)), re-evaluating earlier design choices in terms of QoS, or reconsidering security requirements (i.e. go to step (3)). In some cases, the process may be re-iterated from a new system model or an updated DSSM (restart). Otherwise, a system model with an integrated set of SBBs constitutes a security-enhanced embedded system model.

5.7 Discussions

In this section, we discuss SEED with respect to success indicators formulated within the SecFutur project [17]. Afterwards, we also look at SEED through the lens of software process improvement criteria discussed in research literature.

Success Indicators

To evaluate success of the project 47 success indicators have been formulated by the research and industrial partners [3]. Among which 32 criteria are applicable to parts of the project that are irrelevant in the context of our work. These are, for example, abstract model for embedded systems, implementation of SBBs, code generation, and automated testing. The criteria that we find relevant to the subject of this work belong to two project work packages (WPs): security building blocks WP and security engineering process WP. In total, there are 15 indicators devised for evaluation of

outputs of these WPs. Furthermore, among these 15 indicators we exclude those that analyse the use of the SecFutur approach on concrete showcases and that assess the outputs with respect to other goals of the project. This results in 10 indicators that are suitable for the scope of this work. These are listed below:

- **Indicator 1:** Ease of integration of building blocks.
- **Indicator 2:** Effort saved by using building blocks.
- **Indicator 3:** Suitability of solutions provided by the configuration model and tool.
- **Indicator 4:** Ease of use of the configuration model and tool.
- **Indicator 5:** Ease of integration of the security-aware process.
- **Indicator 6:** Compatibility of the security-aware process with current development processes.
- **Indicator 7:** Use and usefulness of the modelling formalisms.
- **Indicator 8:** Ease of use of the modelling formalisms.
- **Indicator 9:** Improvements in system modelling by using the modelling formalisms.
- **Indicator 10:** Improvements in security requirement management by using the modelling formalisms.

Now we provide our reflection with respect to these indicators.

Indicator 1: The integration aspect is addressed in SEED by promoting usage of such modelling languages (at the realisation level) that support compositional semantics. For example, we adopt the SPACE language that enables verifying that properties of the system are preserved after integration of building blocks. Another viable technique is aspect-oriented modelling where security mechanisms are represented as security aspects. In addition to composability, the SecFutur project proposes to augment each SBB with a special form of security patterns that contains additional integration information.

Indicator 2: It goes without saying that use of pre-defined, already tested, studied, and packaged in a form of SBBs solutions is more efficient than creation of such solutions from scratch. This idea fundamentally lies within component-based development, aspect-based development, and security patterns.

Indicators 3 and 4: The configuration aspects are addressed in SEED in two ways. First, search for a set of SBBs is done based on present assets and corresponding security properties. This step exhaustively explores all possible combinations of SBBs. In such a way, any SBB that is suitable from

the security properties standpoint is shown to a system engineer as a feasible alternative. To limit the set of considered SBBs and to exclude unsuitable alternatives we incorporated the resource criterion into SEED and also have developed the compatibility analysis technique. Here, we must acknowledge that these techniques depend on richness of the repository and quality of the stored information. The development of finely tuned tools was not possible within the scope of thesis work. We provide the proof-of-concept versions. Ease of use is a topic that include such complex aspects as human-machine interaction and graphical design of user interfaces that are not the focus of our work.

Indicators 5 and 6: We designed SEED as an adjunct process to current practices. In particular, from the system design perspectives it adds one activity and does not affect the rest of the practices and processes. Moreover, SEED uses the artefacts produced by a conventional design process – model of system functionality, its execution platform, and allocation – as prescribed by the foundation level. Consequently, SEED is compatible only with those engineering processes where a system engineer models a system (its functional part and execution platform). For a process on the security expert side, SEED only adds the documentation and modelling routines in some cases. One can argue that this can be cumbersome and brings additional complexity. However, it can be addressed by adopting the proposed modelling artefacts (DSSM and PER), and by providing user-friendly tools. This effort, in turn, will be paid off by bringing security concerns close to system engineers.

Indicators 7 and 8: SEED starts when system modelling is already done and does not force a system engineer to adopt new techniques for this task. One can argue that SEED sets extra requirements on system modelling since the quality of SEED outcomes depends on completeness of system models. This is a viable comment. However, we also must acknowledge the added value of modelling in general: proper modelling will also contribute to other phases of system development benefiting from model-driven engineering techniques. For example, appropriate models are suitable for code generation and testing. The modelling required by SEED from security experts has simple syntax and semantics, and is more of the documentation nature.

Indicator 9: The SEED foundation is agnostic to modelling languages and the SEED realisation is adaptive to languages that are inherent to an application domain. Therefore, improvement of system modelling formalisms and languages is an orthogonal issue.

Indicator 10: SEED provides the methods for asset elicitation and for extraction of associated security properties. These two techniques systematically analyse the system model and match it with stored security knowledge. This supports elicitation of security requirements since adequate security properties are pulled from the repository that can be further processed and serve as basics for generation of security requirements. Al-

ternatively, a system engineer with assistance of a security expert would manually go through this process. Hence, we can conclude that SEED contributes to security requirement management by facilitating elicitation of requirements.

Process Improvement

Software process improvement (SPI) is a systematic approach to increase the efficiency and effectiveness of development and to enhance software products [103]. SEED is constructed to enhance the conventional design process by extending it with methods and tools to support security analysis. Hence, SEED can be considered as a process improvement. In this vein, we examine SEED with respect to known factors influencing adoption of development processes.

Baddoo and Hall [104], and Niazi et al. [105] report that the human related factors are among critical success factors for adoption of process improvements. These are, for example, management commitment and staff involvement according to Niazi et al. [105]. Baddoo and Hall [104] mention such human factors as inadequate communication, lack of management direction and commitment, personality clashes, etc. Besides, Baddoo and Hall [104] include factors related to SPI strategy itself, e.g. lack of SPI management skills, commercial pressure, lack of overall support, etc.

Even though we do not focus on detail investigation of such factors, we must mention that security has obviously become an intricate and serious issue for the last two decades since both threats and regulatory pressure on organisations have evolved. Simple threats in a form of viruses distributed through floppy disks have evolved in sophisticated botnets spreading over the network. An adversary model moved from script kiddies to organised criminals that hunt cyber assets. As a result, the burst of attacks against embedded systems, which become known to public, creates significant commercial pressure on organisations. Besides, the regulatory pressure also continues increasing. Thus, a range of standards and guidance have appeared to regulate core industries, e.g. ISO27000-series, HIPAA (healthcare), PCI DSS (payment card industry), NISTIR (smart grids), to name a few. Both threats and regulations stipulate the need to improve current development practices with security related activities.

Niazi et al. [105] mention among major critical barriers that hinder SPI such factors as lack of resources, time pressure, and inexperienced staff. Baddoo and Hall [104] bring such de-motivators as time pressure and constraints, lack of resources, cumbersome processes, lack of evidence of direct benefits, inertia, and inexperienced staff. We proceed with analysing SEED with respect to these factors.

SEED does not intend to change established development processes. In contrast, it is designed to be complementary to existing life cycles, therefore, it does not require additional time and resources to reorganise the whole development routine. The resource- and time-consuming parts of SEED

adoption are maintainability of the repositories and also initial learning of SEED by inexperienced staff.

We envisage that benefits obtained from reusing security knowledge will exceed the costs and resources associated with maintenance of the SEED repositories. The initial learning is facilitated by employing visual modelling and transformation techniques that hide the ontology layers from the users. Thus, inexperience staff needs only a little training about the simple modelling language to be able to create and read DSSMs. Moreover, the SEED realisation relies on modelling languages that are inherent for a certain application domain as opposed to introducing new system modelling languages for enabling security analysis. This prevents unreasonable increase in costs and resources; it also mitigates frustration of inexperience staff who are not forced to learn a completely new language.

Inertia and negative experience criteria is related to unwillingness to adopt a new process if the current one works and when there is no understanding of the purpose of a new process. SEED does not force practitioners to change well-established current processes, but it complements them with security analysis activities. The lack of understanding of the need for (any) security analysis is contracted by the recent uprise of attacks and emerging mindset. A product can not be competitive on the market when it completely neglects basic security issues. Completely unprotected products bear too high risks, especially when these risks could be anticipated with the assistance of such approaches as SEED.

We think that at the initial stage SEED will be more beneficial to such practitioners who do not have any security routines in their design processes at all. In contrast, practitioners who already have security analysis deeply established within their development life-cycles will benefit less. SEED is developed for specialists inexperienced in security and evidence of benefits for such audience lies obviously in incorporation of SBBs into a system. However, we believe that when the SEED repository grows even experienced practitioners will find it useful and profitable.

To conclude, contrary to barriers discussed above, we can distinguish three significant enablers of SEED. These are visual modelling, possibility to use SEED with the existing development processes and modelling languages, and enabled reuse of security knowledge.

6 Quantifying Risks to Data Assets

6.1 Overview

Up till now we have considered that all assets elicited from a system model are candidates for protection that is, in turn, requires integration of SBBs. However, it is often not possible to provide perfect protection for all identified assets. One should be able to sift through only the relevant assets prioritising and trading the potential security provided by integrated SBBs for other economical and resource aspects (as prescribed by step 3 – preliminary analysis of SPs – in Figure 5.17). Therefore, there is a need for a method that will allow quantifying security inherent in a system design.

In this section, we first introduce risks into the picture explaining its main ideas and elements in Section 6.1.1 and Section 6.1.2 followed by examples of application scenarios in Section 6.1.3.

6.1.1 Introducing Risks

Modern system and computing infrastructures are complex artefacts. These systems have a lot of stakeholders whose preferences regarding data assets should be accounted for when deciding the appropriate level of security during design stages. In the telecommunications sector, for example, there is a growing number of end user devices all with their own characteristics (incorporating software and hardware from many different vendors), a number of network operators (wired and wireless), a number of communication system vendors (supporting various access technologies and incarnations of the same standards), as well as regulatory authorities that have the overall national

interests as their domain of interest.

Therefore, when determining which assets and how should be protected a system engineer should have means to answer the following basic questions: Which assets are more important? What assets are more vulnerable to security breaches? Who should compensate for extra costs associated with integrating of SBBs? To be able answering these and similar questions, we need to associate a measure for security with a certain system design. As it naturally comes from the questions being answered, this measure should reflect both the stakeholder take on assets and exposure of these assets to security violation. In this thesis we investigate how the classic notion of risk can be adopted for this purpose.

A security risk is built of two elements: likelihood and consequence. A consequence is an indication of the impact of unwanted incidents on the assets in terms of degree of damage; and a likelihood is the frequency or probability of negative events (unwanted incidents) to occur [106]. Figure 6.1 illustrates how we adopt the notion of risk in our context.

The right hand side of Figure 6.1 visualises the idea that for any given asset different stakeholders may provide different costs for a property violation. While risk analysis tools typically do the weighing and aggregating the various stakeholder perspectives in one model, we extend the domain specific development process to involve different costs associated with different stakeholders. In our work we adopt the view that the notion of cost varies from one application domain to another, from one asset to another, and also from one stakeholder to another. These costs also differ depending on which security goals are violated. For example, for a utility provider as a stakeholder the breach of integrity for user-end electricity measurements are usually associated with high costs, while customer privacy (confidentiality) may have a lower relative priority.

The left hand side of Figure 6.1 shows that the likelihood element is computed using three elements. These are attack models (a negative event), assets (identified by the SEED elicitation technique) coupled with security goals, and also a design model of a system under development itself. A given system design (functional model and selected platform) should be coupled with relevant attack models to obtain the likelihood to compromise a certain security property (that is a triple of asset, security goal, and defence strategy).

Finally, the aggregation of the likelihood and consequence are combined for each asset. The result of this aggregation indicates the risk of losing a certain security goal with respect to an asset. We focus on two basic security goals integrity and confidentiality. Thus, the security metrics defined in this work are intended to describe what are the confidentiality and integrity losses for a system with respect to given attacks. The outcome of the process depicted in Figure 6.1 can be used in step 3 of the SEED process from Figure 5.17 ("Preliminary analysis of SPs to be satisfied") as a means of ranking and filtering the important assets and the less relevant ones.

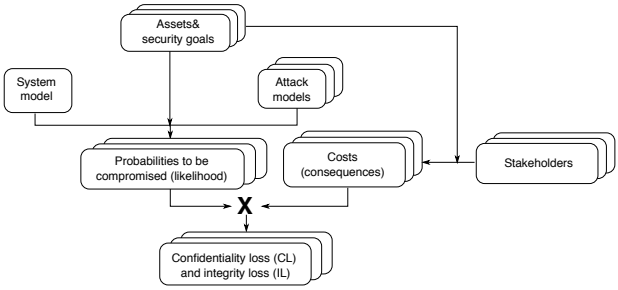


Figure 6.1: Focusing on relevant assets and security goals

6.1.2 Security Goals and Risks to Data Assets

In previous section we have postulated that we focus on confidentiality and integrity losses as risks to data assets. This section motivates these choices.

Confidentiality, integrity, and availability are three basic aspects of security. These three build the *CIA triad* and can be referred to as security properties, goals, aspects, attributes, characteristics, building blocks and so on. Security is concerned about the whole triad, but depending on an application domain (and its stakeholders) one can be prioritised over others. In our work, we provide means to quantify confidentiality and integrity of data assets while omitting availability. To motivate this choice we make a brief review to understand what elements of the system – data, service, or system – can be potentially subject to confidentiality, integrity, and availability losses. Table 6.1 summaries the results of this study.

According to Avizienis et al. [107], security can be represented as built of the three elements of the above triad. Availability is defined as "readiness for correct service" for authorised actions only. Integrity is explained as "absence of improper system alterations" and confidentiality is defined as "the absence of disclosure of information" with respect to unauthorised actions.

Schumacher et al. [108] specify confidentiality as a property that "data is disclosed only as intended by the enterprise"; whereas integrity and availability are discussed with respect to assets, i.e. "integrity is the property that enterprise assets are not altered" and "availability is the property that enterprise assets ... will be accessible when needed for authorised use". Assets include both information and tangible assets such as money.

Jürjens [109] mentions secrecy and integrity with respect to data among important security properties.

Koopman [110] refers to integrity as a property that serves for "ensuring that data or the system has not been tampered with" and to availability as a property that serves for "ensuring that the service provided by the system remains available despite attacks". Confidentiality aspect is mentioned through secrecy and privacy where secrecy is described as "keeping others

from having access to information" and privacy is described as "ensuring that data about a user is not revealed".

Boritz [111] extensively studies information integrity and considers availability as an enabler of integrity.

Parker introduces the Parkerian hexad [112] that is a set of six elements of information security. These are confidentiality, possession or control, integrity, authenticity, availability, and utility. Parker describes availability as "having timely access to information", integrity refers to "being correct or consistent with the intended state of information", and confidentiality is interpreted as "limits on who can get what kind of information".

Ravi et al. [91] mention the CIA triad in the context of functional objectives of attacks against embedded systems. In particular, an objective of an privacy attack is to "gain knowledge of sensitive information", an integrity attack attempts "to change data or code associated with embedded system", and an availability attack wants to "disrupt the normal functioning of the system".

Kocher [9] discusses security requirements for embedded systems. Among these requirements availability is referred to as ensuring that "the system can perform its function and service". Confidentiality and integrity are mentioned in the context of secure communications and secure storage of information.

Fang et al. [113] in their survey on smart grids extensively discuss integrity of data and user privacy where confidentiality of user data is an integral part of it. Similarly, Cleveland [114] focuses on smart grid security challenges and studies confidentiality, integrity, and availability of data.

Besides the research literature the CIA triad is also interpreted by a range of standards. For example, these are NIST 800-27 [115], ISO 27000 [116], and NISTIR7628 [117].

NIST 800-27 [115] determines confidentiality as a security goal that "generates the requirement for protection from intentional or accidental attempts to perform unauthorized data reads". Integrity is defined with respect to both data and system, i.e. "The security goal that generates the requirement for protection against either intentional or accidental attempts to violate data integrity (the property that data has not been altered in an unauthorized manner) or system integrity (the quality that a system has when it performs its intended function in an unimpaired manner, free from unauthorized manipulation)". Finally, the availability is explained as the security goal that "generates the requirement for protection against intentional or accidental attempts to (1) perform unauthorized deletion of data or (2) otherwise cause a denial of service or data".

In ISO 27000 [116] confidentiality is described as a "property that information is not made available or disclosed". Integrity and availability are widely defined and attributed to anything that has value to an organisation. In particular, integrity is defined as a "property of protecting the accuracy and completeness of assets" where an asset can be anything that is valuable

for an organisation; and availability is a "property of being accessible and usable upon demand by an authorized entity".

The guidelines for smart grid cyber security [117] operates with the terms loss of confidentiality, integrity, and availability. Losses of confidentiality and integrity are defined as "the unauthorised disclosure and modification or destruction of information"; and loss of availability is considered to be "the disruption of access to or use of information or an information system".

Table 6.1: Confidentiality, integrity, or availability

	Source	Confidentiality	Integrity	Availability
	Avizienis et al. [107]	information	system	service
	Schumacher et al. [108]	data	asset (data and other tangible assets)	asset (data and other tangible assets)
	Jürjens [109]	data	data	–
	Koopman [110]	information	data and system	service
	Boritz [111]	–	information	—
	Parker [112]	information	information	service (access to information)
	Ravi et al. [91]	data	data and code	system
	Kocher [9]	data	data	system
	Fang et al. [113]	data	data	–
	Cleveland [114]	data	data	data
	NISTIR 7628 [117]	information	information	service (access to data)
	ISO 27000 [116]	information	asset (data, system, and others)	asset (data, system, and others)
	NIST 800-27 [115]	data	data and system	service or data
Total	data(information) service system or code	13 0 0	13 0 5	4 5 3

As it follows from the table, confidentiality is only thought to be applicable to data or information. Integrity is used when referring to both system and data or information, but the latter cases strongly prevail. We can see that availability is also applied to all three categories, but service and system cases outweigh the data category. In the course of our study, we have also observed that the interpretation of the CIA triad is affected by an application domain. For example, we should notice that data avail-

ability is widely applicable in the domain of data storage services, where data availability is often used to characterise the quality of provided services. In critical infrastructures availability is applicable to infrastructure and its services and is one of the main security objectives along with privacy of consumers [117, 113]. We have also seen that in the embedded system domain the data is mainly subject to integrity and confidentiality goals [110, 91, 9, 118].

In our work, we make a first step towards providing a method for quantification of confidentiality and integrity to data assets within system models. Our brief analysis shows that these two considered attributes are well represented when speaking about data assets. We leave the availability goal outside the scope of our work since it is most commonly applied as a characteristic for a system and service.

6.1.3 Application Scenarios

In this section, we discuss application of the proposed metrics, i.e. confidentiality loss and integrity loss. In particular, we discuss two application scenarios where the loss metrics can be employed (1) to analyse the effect of integrated SBBs and (2) to compare alternative designs. For illustration purposes, we use the smart metering infrastructure case.

As described in Section 2 the overall specification of this case consists of 7 main scenarios that have a range of diverse security considerations. Consequently, there are many assets identified in these scenarios, e.g. measurements (meter readings), a set of user account data (customer, administrator, operator), a set of certificates (calibration, installation, deinstallation, manufacturer), communication configurations, functional settings, event records, commands, control messages, etc. Additionally, as any large system the metering infrastructure has many stakeholders. We focus on three assets, namely measurements (denoted by A_1), certificates (A_2), and commands (A_3). We also consider three distinct stakeholders, i.e. end users, the utility provider, and the national regulatory agency.

Violation of confidentiality and integrity of these assets has different consequences for different stakeholders. For example, for a utility provider as a stakeholder, breach of the integrity of measurements is usually associated with high costs. A systematic misuse of the metering device can lead to manipulations at large scale and result in economic losses. However, the breach of confidentiality for the same measurement data is of a lower priority. Obviously, the picture is different for the user as a stakeholder. One can consider the national regulatory agency to be mainly interested in the availability dimension of the electricity supply and thereby, seen from that stakeholder's perspective the breach of confidentiality of the measurement data has a lower consequence. On the other hand, a large scale manipulation of the commands issued to the sensor nodes, can be used in a scenario where national security is threatened.

Application of the SEED approach allows systematically identifying the presence of above assets within a system model. The calculated confidentiality or integrity loss for all assets can be organised in a *stakeholder security profile* that shows losses for a stakeholder with respect to each asset. These profiles can be visualised as plots, e.g. as depicted in Figure 6.2. Here, the selected assets are listed along the x-axis, and the y-axis shows the calculated confidentiality loss.

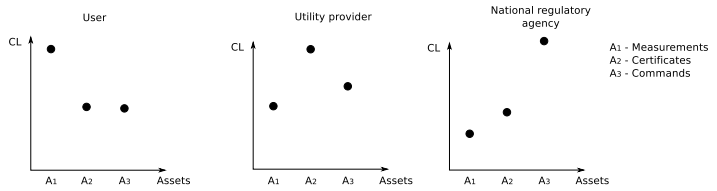


Figure 6.2: Stakeholder security profile view

The goal of the rest of the SEED approach from Figure 5.17 is to select a set of SBBs to reduce potential confidentiality (integrity) loss for one or more stakeholders. Obviously, integration of any new functionality into a system (including security features) will imply extra costs. These costs can be both monetary in terms of added software and hardware components for integrating SBBs, and some reduction of the available computing or memory resources. In order to incorporate the cost of bringing these aspects and potentially to distribute the cost among stakeholders, we need to evaluate how each stakeholder benefits when a certain SBB is integrated. We propose that the added benefit is expressed as a reduction effect [1] that a SBB brings in terms of confidentiality or integrity losses for each asset.

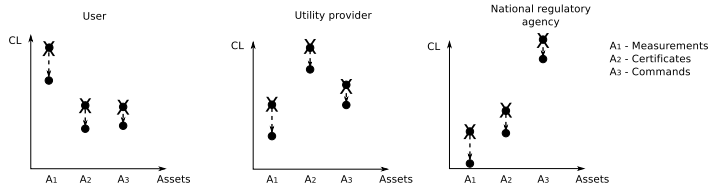


Figure 6.3: Reduction effect of SBBs on stakeholder profiles

For illustration we consider three SBBs from the metering DSSM (depicted in Figure 4.4) selected within the SecFutur project to be integrated into the TSM device. They are secure storage, anomaly detection, and secure communication. Secure storage and security communication reduce the likelihood of breaching integrity and confidentiality of stored data and

[1] The term is inspired by [119]

transmitted data respectively. The anomaly detection [81] aims to reduce the likelihood of integrity loss for measurements stored in the device. Reduction effect of implemented SBBs is visualised in Figure 6.3 as arrows that shift the initial confidentiality loss to lower values[2]. In this way a system designer can analyse which stakeholders benefit most from integration of which SBBs and consider the cost-benefit trade-off for the implementation appropriately.

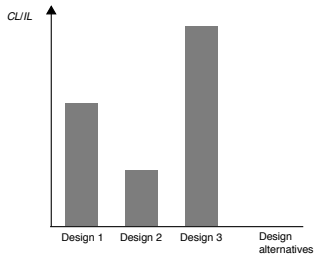


Figure 6.4: Comparison of alternative designs

The proposed risk-based metrics can also be employed to differentiate and compare alternative designs of the same system specifications. Figure 6.4 demonstrates the second application scenario where there are three candidate implementations for a system. Alternative designs can differ in a broad sense. For example, the same metering device can be implemented on different execution platforms that possess different attack surfaces, and thus, implies different risks. Particularities of an internal logic of a system design can also impose different confidentiality and integrity losses, since the way how assets are manipulated by a system can have a big impact on its security properties. For instance, if there are measurements stored on an unprotected memory unit, the way (e.g. frequency) this memory unit is erased naturally influences the probability to success for an attack when an attacker tries to copy the data from this memory unit. This application scenario for the proposed metrics goes beyond the SEED approach, but still a viable case.

In this section we introduced the notion of confidentiality and integrity losses by giving motivation and application scenarios. In the following sections, we provide a complete machinery to support computation of these metrics.

[2]Note that the placement of the dots in the figures and the scale of the reduction (the size of arrows) in the shown diagrams is not the result of exact computations, but only a relative placement to visualise the intended use of the suggested techniques.

6.2 Proposed Metrics

As it is explained in the previous chapter, confidentiality and integrity losses can be naturally defined as risk-based metrics. Risk is typically modelled by the likelihood of an unwanted event and the severity of its consequences. An unwanted event is a mixture of system dynamics and attack behaviour. In particular, different ways of handling assets within a certain system (captured by a design model) will imply different exposure of these assets to confidentiality and integrity breaches that are, in turn, associated with a certain attack vector. In the same vein with SEED that follows the separation of concerns principle, we suggest that security analysis should treat attack and system behaviours as two separate, though interwoven, elements. Both elements are usually highly complex constituent of system security and should be regarded separately for their more accurate elaboration, hence 'separate'; while both of them are clearly interdependent, hence 'interwoven', for example, behaviour of an attack usually depends among other factors on a system design and its reactions.

We also posit that attack and system behaviours should be modelled as time-dependent probabilistic processes. The presence of the time dimension allows accounting for dynamic aspects of potential attacks and a considered system: the probability of a successful attack may change as time progresses, and a system may possess different valuable data assets as its execution unfolds. The use of probabilistic modelling, in turn, enables dealing with uncertainties (both aleatory and epistemic [120]) that are naturally present at the design phase.

One can potentially argue about difficulties of obtaining realistic data about the timing aspects of an attack and system at the design phase, and therefore, question reliability of results of the proposed security analysis. We nonetheless propose that an easier and more effective explorations of security threats and impacts is already a valuable input to design decisions, even when subject to some uncertainties. This enables 'what if' analysis which allows understanding the sensitivity of the system to potential attacks. Furthermore, the research that enables quantitative estimations of timing aspects of attacks and system at earlier design stages constantly progresses.

6.2.1 Confidentiality Loss and Integrity Loss

We define *confidentiality loss* (CL) of a valuable data asset o given an attack A by time t as a risk metric that characterises the damage potentially caused by the attack A to the asset o. It is calculated as a product of the likelihood that the attack A would disclose the asset by t and the cost of this breach for a stakeholder R. In turn, confidentiality loss of a system Y is a function (denoted by the symbol $\otimes$) of confidentiality losses for each data asset o_i that is subject to an attack A. The actual function will depend on the

properties of the data assets in question and stakeholder's take on them.

$$CL(Y, A, R, t) = \otimes_i CL(o_i, A, R, t) \tag{6.1}$$

Similarly, *integrity loss* (*IL*) of a data asset o given an attack A by time t is a risk metric that characterises the effect from the potential alteration of the affected data asset. The notion is analogously extended to the system level.

In the rest of this section, we focus on confidentiality and integrity losses (*CL* and *IL*) for a single asset. Section 6.4 takes this further and discusses extension of the defined metrics to a system level.

6.2.2 Basic Terms: Domain, Attack, and System

In accordance with the domain specialisation principle adopted by SEED, we use an idea of domain as a basic notion that creates a common ground for system engineers and security experts. More specifically, we say that a security expert and a system engineer work in the same application domain when they refer to a common set of components and objects while modelling respectively a system and the attacks. This, in turn, provides us with a mechanism for treating the system and attack as separate though interwoven artefacts.

Definition 4 *A* domain M *is a tuple* (C, O) *where* C *is a set of components and* O *is a set of data objects accessible in an application area. A set of assets is a subset of* O *denoted by* $Assets \subseteq O$.

Attack modelling is a commonly used technique to capture behaviour of attacks. Attack trees or attack graphs are two main examples of such techniques. The basic elements of attack trees and attack graphs are attack steps and relations on them. Attack trees, additionally, have special elements such as gates, which are logical operations applied on attack steps (e.g. AND and OR), and root, which represents the goal of an attack. Quantitative aspects of attack models are captured in several ways. One of them is assigning a probability distribution of execution time to the attack steps. Kordy et al. [121] provide a comprehensive survey of different forms of attack trees and attack graphs. In our work, we use the term *basic attack model* to describe an attack that generalises attack trees and graphs.

Definition 5 *A* basic attack model *is a tuple* (AS, AR, l_{AS}) *where:* AS *is a finite set of attack steps;* $AR \subseteq AS \times AS$ *is a relation between attack steps; and* $l_{AS} : AS \to \mathcal{F}$ *is a labelling function that associates execution time distributions from the set* $\mathcal{F}$ *to attack steps (*AS*).*

We extend this basic definition of an attack model with the *attack step annotation* concept. It enriches the definition of a basic attack model with *what*, *where*, and *how* information: *what* assets are targeted; *where* in a

system (i.e. on which parts of a system platform); and *how* these assets are compromised, meaning which security attributes are violated.

Definition 6 *An* attack step annotation *is a tuple (TA, TC, AV) where:*

- $TA \in 2^O$ *is a set of* ***t****argeted* ***a****ssets;*
- $TC \in 2^C$ *is a set of* ***t****argeted* ***c****omponents;*
- $AV \in 2^{Attr}$ *is a set of security* ***a****ttributes* ***v****iolated by the attack step where* $Attr = \{Cnf, Int, Avl\}$ *(Confidentiality, Integrity, Availability).*

We denote a set of such annotations by N and we refer to each element x of the attack step annotation as by as.x (e.g. as.TA).

For example, if an attack step reads message $m \in O$ from a component $link \in C$ then an AS annotation for this attack step is $(m, link, Cnf)$[3]; if an attack step only connects to some $link \in C$ then its annotation is $(\emptyset, link, \emptyset)$; if an attack step changes some file $f \in O$ stored on the device memory unit $u \in C$ then its annotation looks like (f, u, Int). These annotations allow relating an attack model to relevant elements of a system model. This, in turn, enables combining attack models with system models. A basic attack model enriched with annotations is called an *annotated attack model*.

Definition 7 *An* annotated attack model *A is a tuple* (AS, AR, l_{AS}, l_N) *where* (AS, AR, l_{AS}) *is a basic attack model and* $l_N : AS \to N$ *is a labelling function that assigns an annotation to each attack step.*

Next, we present the elements of a system that we need to capture in our formalisation of a system model. For our analysis, we need to capture two aspects of a system: its functionality, execution platform and allocation information, and information about data object dependencies. These two aspects are represented by a *state model* and a *data model*.

Definition 8 *A* state model *SM of a system is a tuple* (S, s_0, P, H, l_O, l_S) *where:*

- S *is the set of system states related to each other by a set of transitions;*
- s_0 *is the initial state;*
- $P : S \times S \to [0, 1]$ *associates a probability with a transition;*
- $H : S \to \mathcal{F}$ *associates a probability distribution with a state;*
- $l_O : S \to 2^O$ *is a labelling function that associates a set of objects from domain M with each state;*

[3]To simplify the representation of AS annotations we omit the curly brackets when denoting a set that is built of one element.

- $l_C : S \to 2^C$ *is a labelling function that associates a set of components from domain* M *with each state.*

A state in S can be seen as a basic element of behaviour (e.g. an action) of a system. P and H formalise the probability of moving from one state to another and the probabilistic measure of execution time of each system state respectively. Thus, the first four elements of our state model form a semi-Markov chain [122] (SMC). The latter two elements extend a SMC with additional information that is utilised to automatically combine system and attack models.

Function l_O allows capturing the information about data objects (including assets) which exist at a certain state. Function l_C associates the states with components of an execution platform.

Definition 9 *A* data model DM *of a system is a tuple* (D, l_D) *where:*

- $D \subseteq O \times O$ *is a relation that captures immediate object dependency;*
- $l_D : D \to 2^S \setminus \emptyset$ *is a labelling function that associates a set of states from* S *with each tuple in* D*.*

The relation D represents dependencies between data objects in an analysed system. In particular, $(o_i, o_j) \in D$ means that an asset o_j depends on an asset o_i; for example, $o_j = f(o_i)$ where f is some function. We omit the nature and strength of such dependencies, however, this information can also be utilised. The function l_D captures at which system state the dependencies in D occur. Thus, if $l_D(o_i, o_j)$ returns state s then it means that o_j is derived from o_i in s. Implicitly, a well-formed data model associates a non-empty state set to every element in D.

Finally, a system is a tuple of a system model and a data model.

Definition 10 *A* system model Y *is a tuple* (SM, DM) *where SM is a system model and DM is a data model.*

We summarise the introduced notation and terms in Table 6.2.

6.2.3 Metrics and Their Derivation

We now go on to define the *CL* and *IL* metrics and show how they are derived based on the formalised above system and attack models.

Confidentiality loss

Recall that *confidentiality loss* (*CL*) caused by an attack A to each valuable data asset o by time t is a product of the likelihood that A would disclose o by t, and the cost of this disclosure to stakeholder R. In our context, the likelihood is a probability. Thus,

Table 6.2: Summary of the used notation

Sets and subsets		
C – components	AV – security **a**ttributes **v**iolated	
O – objects	N – attack step annotations	
$Assets$ – assets, $Assets \subseteq O$	S – system states	
$\mathcal{F}$ – probability distributions	$\Omega_{o,o'}$ – all state sequences between o' and o	
AS – attack steps	$S_{\langle o \rangle}$ – system states where object o exists	
TA – **t**argeted **a**ssets	C_{target} – system components targeted by an attack	
TC – **t**argeted **c**omponents	$AS_{\langle Cnf,o \rangle}$ – attack steps violating confidentiality of an object o	
Functions, dependencies and relations		
l_{AS} – assigns execution time probability distributions to attack steps, $l_{AS} : AS \to \mathcal{F}$		
l_N – assigns an annotation to an attack step, $l_{ASN} : AS \to \text{ASN}$		
P – associates a probability with a transition, $P : S \times S \to [0, 1]$		
D – a dependency relation between data objects, $D \subseteq O \times O$		
l_D – associates a set of system states with a data dependency, $l_D : D \to 2^S$		
H – associates a probability distribution of execution time with a state, $H : S \to \mathcal{F}$		
l_O – associates a set of existing objects with a state, $l_O : S \to 2^O$		
l_C – associates a set of components with a system state, $l_C : S \to 2^C$		
AR – a relation between attack steps, $AR \subseteq AS \times AS$		
$cost$ – a cost of asset disclosure or alternation expressed by a stakeholder		
κ – a function that checks whether there is a transitive dependency between two objects		
Tuples		
$SM = (S, s_0, P, H, l_O, l_C)$ – a state model	$M = (C, O)$ – an application domain	
$DM = (D, l_D)$ – a data model	$Y = (SM, DM)$ – a system model	
$A = (AS, AR, l_{AS}, l_{ASN})$ – an annotated attack model		
Other		
R – a stakeholder	PE – propagation effect	ω – sequence of states
CL – confidentiality loss	DE – direct effect	γ – sequence of data objects
IL – integrity loss	ϕ – interval transition probability	

$$CL(o, A, Y, R, t) = p(o, A, Y, t)\ cost(o, R) \tag{6.2}$$

In equation (6.2), the cost of an asset, $cost(o, R)$, is a subjective estimate expressed by a stakeholder R. In general case, the cost can also be time-dependent, but in this work we assume that it is time-agnostic. In turn, a probability of disclosure, $p(o, A, Y, t)$, can be broken down into a product of two independent events: (E_1) an attack A is in a step that can disclose o by time t; and (E_2) an asset o actually exists in system Y when it is attacked by time t.

$$p(o, A, Y, t) = p(E_1(o, A), t)\ p(E_2(o, Y), t) \tag{6.3}$$

To put it another way, the E_1 event accounts for a subset of attack steps $AS_{\langle Cnf,o \rangle} \subseteq AS$ that compromise an asset o and violate its confidentiality; and the E_2 event accounts for a subset of system states $S_{\langle o \rangle} \subseteq S$ that are associated with asset o. Additionally, the attack steps from $AS_{\langle Cnf,o \rangle}$ should target a set of components that are used for allocation of system states from $S_{\langle o \rangle}$. This simply means that, for the attack to be successful, a system should have components with certain targeted vulnerabilities exploited by the attack steps from $AS_{\langle Cnf,o \rangle}$. We refer to this subset of targeted components as C_{target}.

Given a set of states S from a system Y and a set of attack steps AS from an attack A the set of targeted components C_{target} is defined as follows:

$$C_{target} = \{c \mid s \in S, as \in AS, c \in l_C(s) \cap as.TC\} \tag{6.4}$$

Given a set of attack steps AS and a set C_{target} then a subset of attack steps that disclose an object o is defined as follows:

$$AS_{\langle Cnf,o\rangle} = \{as \mid as \in AS, as.TC \cap C_{target} \neq \emptyset, o \in as.TA, Cnf \in as.AV\} \tag{6.5}$$

Given a set of system states S and a set of targeted components C_{target} then a set of states where an object o potentially can be comprised is defined as follows:

$$S_{\langle o\rangle} = \{s \mid s \in S, l_C(s) \cap C_{target} \neq \emptyset, o \in l_O(s)\} \tag{6.6}$$

In other words, execution of any attack step in $AS_{\langle Cnf,o\rangle}$ leads to disclosure of a given object o, which is essentially the E_1 event. This corresponds to construction of an attack tree with attack steps from $AS_{\langle Cnf,o\rangle}$ which are all connected by the *OR* gate. Thus, the probability that an attack A discloses o in a system Y by time t can be computed as follows:

$$p(E_1(o, A), t) = 1 - \prod_{as \in AS_{\langle Cnf,o\rangle}} \Big(1 - p(as, t)\Big) \tag{6.7}$$

Finally, given the SMC that underlies the system state model, the probability that an asset o exists in Y by time t can be computed as follows:

$$p(E_2(o, Y), t) = \sum_{s \in S_{\langle o\rangle}} \phi(s_0, s, t) \tag{6.8}$$

In equation (6.7), $p(as, t)$ is a probability of success of an attack step as by time t within an attack model A. It is returned by l_{AS} given t that is, in turn, calculated from a selected modelling formalism for attack step relations (e.g. attack trees or graphs). In equation (6.8), $\phi(s_0, s, t)$ is a so called interval transition probability of the system Y transiting from a state s_0 to a state s in interval $(0, t)$ [122]. It is calculated from the system equation shown in Section 2 that describes the dynamics of a SMC.

Integrity loss

Integrity loss (IL) is a metric that defines the risk of alterations to an asset o, and should account for two aspects:

- the loss caused by the direct influence of an attack A on asset o, referred to as the *direct effect* (DE);

- the loss caused by spreading corrupted data and further contaminating the computations dependent on asset o, referred to as the *propagation effect* (PE).

Hence, integrity loss is built of two compounds:

$$IL(o, A, Y, R, t) = DE(o, A, Y, R, t) + PE(o, A, Y, R, t) \tag{6.9}$$

The reason to include the propagation effect in the IL metric, but not in the CL metric can be explained with the following rationale. Whether a breach of confidentiality will propagate depends on specific attack capabilities, i.e. on whether an attack is capable of learning additional data when it has observed a part of it. This act of data reconstruction and learning is actually a self-contained attack step and should be explicitly included into an attack model. For example, the sole fact that an attack compromises an encryption key does not directly implies that all data encrypted by this key is compromised. It is compromised if an attack actually captures and reads the data protected with this key. This behaviour and dependency should be explicitly modelled as part of an attack model. In contrast, a breach of integrity of some data will propagate independently on a considered attack, but depends on the attacked system behaviour, i.e. how a system further uses the corrupted object. For example, if a public key is modified, then all data signed by this compromised key can not be decrypted if the decryption state is also part of a system.

The direct effect DE is calculated in analogy to CL, where $AS_{\langle Int,o\rangle}$ is defined similarly to $AS_{\langle Cnf,o\rangle}$, but $Int \in as.AV$ replaces the corresponding term in equation (6.5).

The intuition for the propagation effect is as follows: if an object $o' \in O$ is computed in a state $s' \in S$ based on an object o that has already been corrupted in a state $s \in S_{\langle o\rangle}$ then o' is also considered corrupted. To derive this propagation effect PE with respect to each such object o' we need to consider the four aspects that make up elements of equation (6.10) below.

First, we need to check whether o' immediately or transitively depends on o. The immediate dependency is already captured in data model DM by $(o, o') \in D$. We say that transitive dependency exists, if it is possible to construct such a sequence of objects γ of length n that $\gamma = (\gamma_k \mid \gamma_1 = o, \gamma_n = o', 1 \leq k < n : (\gamma_k, \gamma_{k+1}) \in D)$[4]. We formalise this test as a function $\kappa : O \times O \to \{0, 1\}$ that returns 1 if such a sequence γ exists, otherwise it returns 0.

$$\kappa(o, o') = \begin{cases} 1, & \text{if } \exists \gamma = (\gamma_k \mid \gamma_1 = o, \gamma_n = o', 1 \leq k < n : (\gamma_k, \gamma_{k+1}) \in D) \\ 0, & \text{otherwise} \end{cases}$$

[4]We use the subscript notation, i.e. γ_i, to access the i^{th} element of the sequence γ.

The next two elements are the cost of o' as expressed by a stakeholder R and the probability that o will be actually attacked by some time $\tau \leq t$ in the first place.

Finally, the propagation effect occurs only when the system Y will transit from a state $s \in S_{\langle o \rangle}$ to a state s' where o' is computed from o immediately or transitively. Such a state s' can be returned by the labelling function l_D, if immediate dependency between o and o' exists. However, if an immediate dependency does not exist, but a transitive dependency exists then we need to consider a sequence of states ω (of length $n-1$) along which the transitive dependency, captured by a sequence of objects γ (of length n), occurs. We construct ω in such a way that $\omega = (\omega_k \mid 1 \leq k < n-1 : \omega_k \in l_D((\gamma_k, \gamma_{k+1})))$. Since there can be several valid sequences of states relating o and o', we denote by $\Omega_{o,o'}$ a set of such state sequences. In other words, $\Omega_{o,o'}$ is the set of all state sequences along which o' can be compromised when o is attacked.

The propagation effect given the four elements described above is calculated as follows:

$$PE(o, A, Y, R, t) = \sum_{o' \in O} \kappa(o, o')\ cost(o', R) \sum_{s \in S_{\langle o \rangle}} p(E_1(o, A), \tau) \sum_{\omega \in \Omega_{o,o'}} P(s_0, s, \omega, t) \tag{6.10}$$

In equation (6.10), $P(s_0, s, \omega, t)$ is the interval transition probability that the system that starts at s_0 will first pass the state s (where asset o is attacked by A), and then will go through each state from the sequence ω. This probability can be computed recursively as follows:

$$P(s_0, s, \omega, t) = \phi(s_0, s, \tau)\ P(s, \omega_1, \omega_{[2..]}, t - \tau) \tag{6.11}$$

We denote by ω_1 the first element of the sequence ω and by $\omega_{[2..]}$ a suffix of this sequence ω starting from the 2^{nd} element. The validity of equation (6.12) can be proven by simple induction.

Theorem 1 *Given a system Y, its two distinct states s_0 and s, and a sequence of states ω^i of length i then the interval transition probability $P(s_0, s, \omega^i, t)$ that the system starts at s_0 will, first, pass the state s, and then will go through each state from the sequence ω^i can be computed as follows:*

$$P(s_0, s, \omega^i, t) = \phi(s_0, s, \tau)\ P(s, \omega^i_1, \omega^i_{[2..]}, t - \tau) \tag{6.12}$$

Proof 1 We show this by induction over i.

For the **basic step** we consider a sequence that has only one element, i.e. $\omega^1 = (s_j)$. Equation (6.12) applied for this case gives the following expression:

$$P(s_0, s, s_j, t) = \phi(s_0, s, \tau)\ P(s, s_j, \emptyset, t - \tau) \tag{6.13}$$

In equation (6.13), $P(s, s_j, \emptyset, t-\tau)$ can be interpreted as the interval transition probability that the system starts at s and enters state s_j. This, in turn, is a simple interval transition probability $\phi(s, s_j, t-\tau)$. Thus, we should show that

$$P(s_0, s, s_j, t) = \phi(s_0, s, \tau)\ \phi(s, s_j, t-\tau) \tag{6.14}$$

To show validity of equation (6.14) we interpret $P(s_0, s, s_j, t)$ as occurrence of two events: (e_1) the SMC reaches state s by some time point τ given that it enters state s_0 at time 0; (e_2) the SMC reaches state s_j by some time point t given that it enters state s at time τ. Thus, $P(s_0, s, s_j, t) = p(e_1, e_2)$. Next, $p(e_1, e_2)$ can be expressed through conditional probabilities as

$$p(e_1, e_2) = p(e_2 \mid e_1)\ p(e_1) \tag{6.15}$$

In equation (6.15), $p(e_2 \mid e_1)$ means that state s_j is reached when the SMC enters s at time $\tau < t$. That is, in turn, is equal to standard SMC interval transition probability $\phi(s, s_j, t-\tau)$ where we start process in state s at time 0. Simply by construction, $p(e_1)$ is the SMC interval transition probability $\phi(s_0, s, \tau)$. Thus,

$$p(e_2 \mid e_1)\ p(e_1) = \phi(s, s_j, t-\tau)\ \phi(s_0, s, \tau) \tag{6.16}$$

Obviously, equations (6.16) and (6.14) are identical.

For the **inductive step** we assume that equation (6.12) holds for a sequence of length n, i.e. $\omega^n = (s_1, s_2, ..., s_n)$.

Now we need to show that equation (6.12) holds for a sequence of length $n+1$, i.e. for $\omega^{n+1} = (s_1, s_2, ..., s_n, s_{n+1})$. We write the equation for ω^{n+1} as follows:

$$P(s_0, s, \omega^{n+1}, t) = \phi(s_0, s, \tau)\ P(s, \omega_1^{n+1}, \omega_{[2..]}^{n+1}, t-\tau) \tag{6.17}$$

where in equation (6.17) we have

$$P(s, \omega_1^{n+1}, \omega_{[2..]}^{n+1}, t-\tau) = P(s, s_1, \omega^n, t-\tau) \tag{6.18}$$

As it comes from our inductive assumption, equation (6.12) holds for the right part of equation (6.18). Thus, equation (6.12) obviously holds for equation (6.17). □

6.3 Application to Smart Meter

In this section we apply our methodology on an open platform device developed based on the design from the European project SecFutur [17], where a Trusted Sensor Network (TSN) was a case study (described in Section 2). We illustrate the application of our methodology and show how confidentiality and integrity losses capture the security risks associated with data assets.

The TSN is built of a set of metering devices, database servers, client applications, and a communication infrastructure. Recall that the main goal of this system is to measure energy consumption at households and to associate measurements with the clients' data for billing purposes. The overall specification of this infrastructure consists of 7 main scenarios that have a range of diverse security considerations [3]. Consequently, there are many assets identified in these scenarios, e.g. measurements (meter readings), a set of user account data (customer, administrator, operator), a set of certificates (calibration, installation, manufacturer), communication configurations, functional settings, event records, commands, control and command messages, etc. In this section, we study a *metering device* and focus on *measurements* as an asset.

We consider three stakeholders: user, utility provider, and national regulatory agency. The stakeholder costs of losing confidentiality or integrity for measurements are shown in Table 6.3. To capture stakeholder estimations we adopt a common linguistic scale [123] {*insignificant, minor, moderate, major, catastrophic*} to encode these costs, which we further map to a numerical vector $\{0, 0.1, 0.5, 0.8, 1\}$ where 0 means "no cost" and 1 means "extremely high costs". Note that we use this simplified numerical scale only for exemplification. In practice, one would use intervals or even continuous scales. Moreover, the selected linguistic scale can also change from one application domain to another. CORAS suggests deciding on such scales as a result of step 4 for each risk assessment. Park et al. [124] introduce an algorithm to automatically score and rank assets by estimating their criticality for the business and organisation. Our methodology does not impose specific requirements on this form of the scale, and therefore, it can be adapted for a particular case.

Table 6.3: Stakeholder costs expressed for measurements

	User	**Utility provider**	**National agency**
Confidentiality	major	minor	insignificant
Integrity	moderate	major	minor

6.3.1 System Modelling

Figure 6.5 depicts a system model for a metering device used in the TSN scenario. This functional model is expressed as an UML activity diagram. The system expects a command (*cmd*) from an administrator of a utility company. On receipt, it proceeds to registration, configuration, or calibration procedures. When the device is calibrated it starts collecting raw data (*raw_msr*), processing them into ready measurements (*msr*), writing them into the memory and creating a unique id (*id_msr*), sending them to

the server, and waiting for an acknowledgement (*ack*). If an acknowledgement has not arrived a meter continues to collect measurements; otherwise, it proceeds to the verification procedure. If verification succeeds the device searches for the measurement by id (*id_msr*), deletes this measurement from storage, and waits for the next command from the server. If verification fails, the device reads the measurement from storage and resends it.

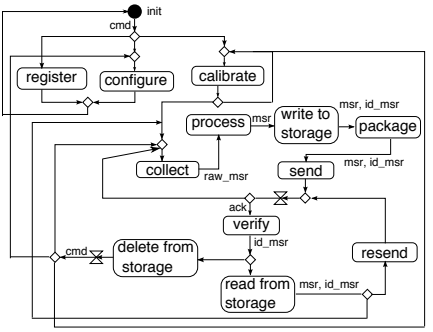
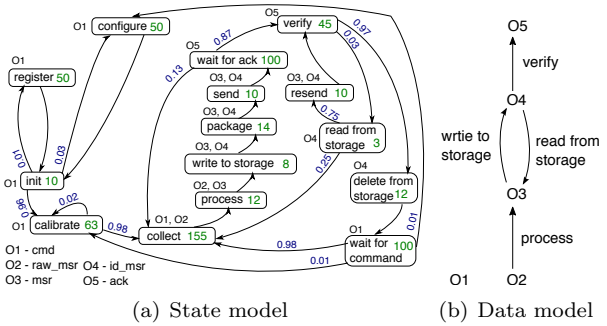


Figure 6.5: The metering device functional model

Construction of state and data models (introduced in Section 6.2.2) can be accomplished by traversing and transforming control and data flows of the UML activity model. UML activity diagrams can be directly transformed into a state model [125]. Alternatively, UML activities can be first transformed into some variant of Stochastic Petri Nets (SPNs) [126]. They offer easier translation from UML activity diagrams and also provide great capabilities such as dealing with concurrency and various forms of dependency. These variants of SPNs can be further used to generate Markov and none-Markov models, i.e. SMC in our case. For the purpose of this illustration we directly move on to the state model and omit intermediate steps (the used semantic function is explained in Appendix C). A data model can be obtained by traversing the UML activity and utilising its built-in data flows. Note that we use the UML activity as an example. Our approach is not limited to this choice.

Figure 6.6(a) depicts the state model (SM). The numbers on arcs are probabilities of the transitions (P) and the numbers on states are mean state execution times of normal distributions (H). To obtain this data, design phase estimation methods exist. We employ prototyping for execution time estimates as a viable alternative. Details of the designed scenario are described in Appendix B. The object references next to each state correspond to the labelling function l_O. Our metering device model includes a simplified execution platform built of two components: a *link* and a *device*. The *link* is a communication network, and the *device* is a meter itself. The *init*, *package*, *send*, *wait for acknowledgement*, *resend*, and *wait for command* states are allocated on the *link*, and the rest of the states are executed on the

device. This corresponds to the labelling function l_C. Figure 6.6(b) depicts the data model (DM), where the labels on dependency arcs are names of corresponding states from SM. This corresponds to the labelling function l_D.



(a) State model (b) Data model

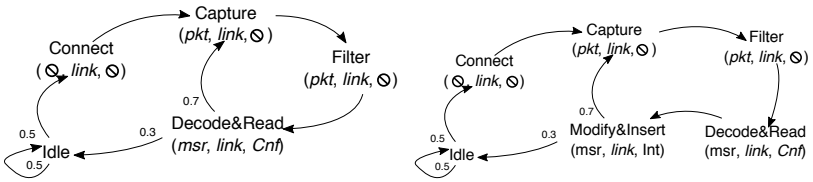
Figure 6.6: System model for the metering device

6.3.2 Attack Modelling

Privacy and integrity of measurements are two serious concerns for smart metering devices [127, 128, 113]. Privacy can be violated by eavesdropping energy consumption measurements that are passed over the communication network used by meters. By collecting the meter readings an attacker can reveal personal information, e.g. individual's behaviour, habits, activities, and preferences [129, 130]. Integrity of measurements can be broken by modifying the original data sent by a metering device. By fabricating energy meter readings an attacker can manipulate energy costs, or even sabotage operation of a control center that rely on accurate measurements for estimating the power grid state [113]. In this section, we consider these two attacks, i.e. eavesdropping of measurements and measurements spoofing, as examples. Respective annotated attack models in the form of attack graphs are depicted in Figures 6.7(a) and 6.7(b).

To conduct the eavesdropping attack an attack should first succeed to sniff the data packets (*pkt*) sent over the media (*link*). This, in turn, requires that an attack connects to the media and captures packets. Thereafter, the attack filters required packets, and, finally, decodes and reads them. Then, the attack proceeds capturing the next packet or it goes into the idle state (inactive state for an attack) where it can start over. Similar to the eavesdropping attack, the measurements spoofing attack starts with intercepting, decoding, and reading the required packets. Thereafter, it modifies the measurements and inserts them back into the used media. The attack step annotations in Figures 6.7(a) and 6.7(b) follow the syntax introduced

in Section 6.2.2. The idle state in both attacks allows us to model sporadic presence of an attack in the deployment environment.



(a) Measurement eavesdropping (b) Measurement spoofing

Figure 6.7: Two attacks against measurements

In addition to annotations, each attack step is associated with a probability distribution (the labelling function l_{AS}). Arnold et al. [131] discuss that there are two alternative approaches to obtain such distributions: (1) based on historical or empirical data; and (2) based on expert opinions. For our illustrative example, we employ the former approach by running experiments with the earlier constructed prototype. We use the kernel density estimations [132] to obtain the needed distribution functions from these samples.

6.3.3 Calculating Metrics

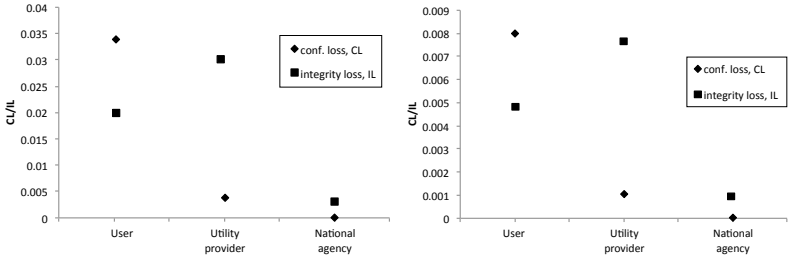
In this section, we show *CL* and *IL* metrics calculated for measurements (*mrs*) as a data asset. To solve SMC models, we use the PRISM model checker [133]. Even though PRISM does not allow direct modelling of SMCs, we have implemented transformation that approximates a SMC as a DTMC (discrete-time markov chain) which analysis is supported by PRISM. The details of this transformation are discussed in Appendix A. We have developed a Python script to support this activity. The calculations of *CL* and *IL* are also supported by a Python script.

A composition of the attack and system models gives the following sets of compromising attack steps (used in equation 6.7) and system states (used in equation 6.8) introduced in Section 6.2.3:

$$AS_{\langle Cnf,msr\rangle} = \{Decode\&Read\ packets\}$$
$$AS_{\langle Int,msr\rangle} = \{Modify\&Insert\ packets\}$$
$$S_{\langle mrs\rangle} = \{package,\ send,\ resend\}$$

We show the values observed when confidentiality (*CL*) and integrity losses (*IL*) stabilise. In reality the risks can change over time, which is also the case in our illustrative example, but we do not show the whole trend since the risks stabilise relatively quick (see Appendix D for more details).

Figure 6.8(a) depicts the calculated confidentiality and integrity losses of measurements. These figures show that both the user and utility provider stakeholders have noticeable risks associated with *CL* and *IL* for measurements; whereas *CL* and *IL* for a national regulatory agency are much lower. Additionally, the results clearly demonstrate that the users have risks associated with both confidentiality and integrity of measurements; while *IL* of measurements for the utility provider exceeds its *CL*.



(a) Original design (b) Modified design

Figure 6.8: Visualisation of calculated *CL* and *IL* for stakeholders

The confidentiality loss stabilises for the user and utility provider stakeholders at 0.035 and 0.004 respectively; and for national agency it is equal to 0 due to the zero costs expressed by this stakeholder for the measurement asset. Thus, confidentiality loss for the user are about 8 times higher than for the utility provider. The integrity loss stabilises for the user, utility provider, and national agency at 0.02, 0.03, and 0.003 respectively. We can see that the utility provider bears the highest relative *IL*, but that is still comparable to *IL* of the user. The national agency bears the lowest risk both in terms of *CL* and *IL*. This can be logically explained by the fact that a national regulatory agency is mostly concerned about availability of an energy grid rather then about integrity and confidentiality of measurements.

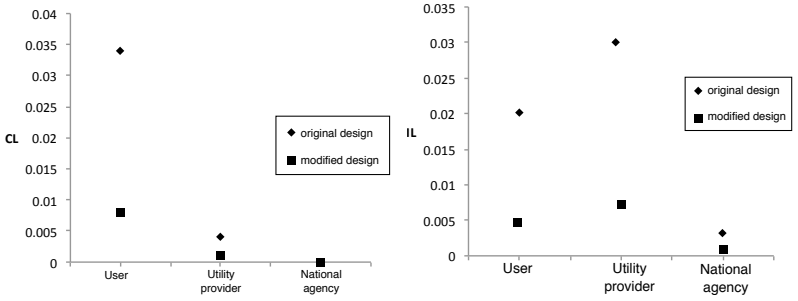
It should be mentioned that due to a narrowed down set of considered assets (i.e. measurements only) in our example, stakeholder-wise comparison of *CL* and *IL* is not as informative as it could be in a general case with multiple assets. In general case, different stakeholders can value different data objects. However, what our example demonstrates distinctly is how the proposed metrics reflect reduction of risks when mitigation measures are applied. That, in turn, indicates how each stakeholder benefits when the original design is modified for strengthening its security aspects [134].

To illustrate our statement, we illustrate how a modification of a design can act as a mitigation against the two attacks. We modify the state "collect", so that the system in Figure 6.6 sends measurements in chunks of 10 readings. By this, the mean execution time of this state ("collect") is changed from 155 to 1550 ms.

Figure 6.8(b) shows results for a modified system, and we observe a significant drop in initial risks (as described shortly). In particular, it shows that *CL* for the user and utility provider converges to 0.008 and 0.001 respectively. Thus, these risks are reduced by factor of 4 in comparison with the risks derived from the original design. Similar, a significant drop is observed for integrity losses, i.e. *IL* for all stakeholders is 3–4 times lower than in the original design.

The cause of the observed risk reductions can be explained by the fact that the probability of measurement existence in a communication channel, which corresponds to the $p(E_1(o, A), t)$ component in equation (6.3), decreases in case of the modified design. Note that the layouts for Figures 6.8(a) and 6.8(b) are similar since the consequence component, which corresponds to $cost(o, R)$ in equation 6.2, is the same for both calculations, while the probability of asset violation is changing due to the introduced modification.

Figure 6.9 visualises the same results, but in a different plane. From this visualisation one can easily compare risks imposed by an original and modified designs with respect to different stakeholders.



(a) Confidentiality loss (b) Integrity loss

Figure 6.9: Alternative view on calculated *CL* and *IL*

Once the risks for the stakeholders are estimated as confidentiality and integrity losses, the next step is evaluation of the obtained risks to make decision. This is an extensive decision problem that typically involves other criteria (e.g. resource footprint of countermeasures, quality of service requirements, etc.). One also could proceed deciding on an acceptance risk level and accept only those design alternatives which risks lie within the acceptance level (as suggested by CORAS).

This demonstrates that there is a potential for visualising and reasoning about the proposed metrics as a design progresses and gets refined. It also shows how simple design decisions can affect the security risks associated with data assets of a system. Thus, our metrics has potential to guide a system engineer by bringing awareness about security to the design phase.

6.4 Extending Losses to System Level

As we have stated in Section 6.2, confidentiality loss of a system Y is a function (denoted by the symbol $\otimes$) of confidentiality losses for each data asset o_i that is subject to an attack A.

$$CL(Y, A, R, t) = \otimes_i CL(o_i, A, R, t) \tag{6.19}$$

The similar statement applies to the integrity loss. We also have mentioned that the actual functions will depend on the properties of the data assets in question and stakeholder's take on them. In this section, we discuss the form of these functions.

To begin with, confidentiality and integrity losses of an asset can be considered as an *expected* loss for stakeholders (a negative variant of an expected value). For example, confidentiality loss of an asset o can be represented as follows:

$$CL(o, A, R, t) = p(o, A, t) \cdot cost(o, R) + (1 - p(o, A, t)) \cdot 0 \tag{6.20}$$

In equation (6.20) the first summand represents the risk of the asset o losing its confidentiality previously presented as equation (6.2). The second summand naturally complements the formula with the "risk" of the asset o not losing its confidentiality. Clearly, the cost of staying uncompromised is equal to 0. Similar reasoning can be applied to integrity loss.

Now, we consider a system Y that has several assets $o_1, o_2, ..., o_n$. Following the reasoning above, we define confidentiality and integrity loss for the system Y as the sum of respective expected losses of each asset in the system.

$$CL(Y, A, R, t) = \sum_{o_i \in Asset} CL(o_i, A, R, t) \tag{6.21}$$

$$IL(Y, A, R, t) = \sum_{o_i \in Asset} IL(o_i, A, R, t) \tag{6.22}$$

The expression in equation (6.21) gives a simple way to extend the metrics proposed in our work to the system level. However, it also sets additional requirements on the cost function that we discuss further.

For example, let us consider a password and a phone number as two ways to authenticate access to some resource. If an attacker modifies only the password or the phone number, an owner still can access the resource by using an unaltered alternative. Therefore, the risks associated with losing one of these assets can be comparatively low. However, when both means are modified by the attacker, the security risks for a stakeholder increase drastically since it becomes impossible to access the resource. This naive

example illustrates that higher or lower risks are tightly connected to the way the costs are estimated by a stakeholder. In particular, the cost of losing a certain asset depends on the context (i.e. the state of and interdependence with other assets) and so does the risk.

This concept can be explained from the utility theory standpoint. Utility is a subjective measure of the amount of satisfaction (or dissatisfaction) a stakeholder would derive. Thus, the amount of dissatisfaction is much higher when both assets (a password and a phone number) are corrupted in comparison with the case when only one of them is corrupted. Moreover, the dissatisfaction associated with corruption of both assets might not be perceived as equal to the sum of the dissatisfactions when each asset is corrupted alone.

To account for such a phenomenon, we suggest to refine the notion of cost used in equations (6.2) and (6.9) for calculating confidentiality and integrity losses. In the rest of this section, we discuss the refined form of this cost function.

The notion of cost used earlier in our work is the cost of losing confidentiality or integrity of a certain asset. We have assumed in the earlier sections, that a cost for a certain asset is independent from other assets. Now we relax this assumption by introducing a cost function $cost'$ to assign the cost of compromising an asset in relation to other assets. This function expresses the cost of losing confidentiality for a certain asset given that none of the other assets is compromised, as well as the cost when any of possible combinations of other assets are compromised. We refer to this cost function as *full cost function*.

For convenience we use the following simplified notation.

- We denote by $cost(o)$ a cost expressed by stakeholder R (we omit R in the notation).
- We denote by $p_o(t)$ a probability that asset o will be compromised by time t given an attack A (we omit A in the notation).
- We denote by $Asset_Y \subseteq Asset$ the set of assets in the system Y.
- We denote by $cost'(o)$ the full cost function of an asset o for stakeholder R (we omit R in the notation).

We now design the *full cost function* that is a refinement of the initial cost function that we have worked with in the earlier sections. We write $cost(o_i \mid o_j)$ to denote "the cost of asset o_i given that o_j has been compromised while other assets from $Asset_Y$ have not been compromised". The cost $cost(o)$ that we have used before should be interpreted as $cost(o \mid \emptyset)$ that is "the cost of asset o when none of the other assets is compromised". It is also clear that $cost(o_i \mid o_j)$ is equal to $cost(o_i)$ when a stakeholder does not see any relation between o_i and o_j in terms of their costs.

Figure 6.10 illustrates relative costs for our example with a password and a phone number for authentication before accessing a resource.

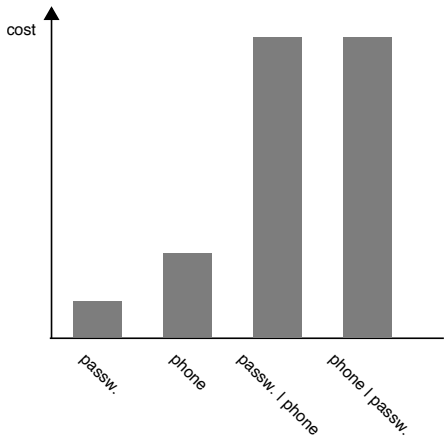


Figure 6.10: An example of cost for compromised assets in the context of other assets

We define the full cost function $cost'$ as follows:

$$cost'(o) = \sum_{x \in 2^{Asset_Y}} w(o \mid x)\ cost(o \mid x) \tag{6.23}$$

Equation (6.23) expresses that the cost of losing confidentiality (integrity) of a certain asset o is equal to a weighted sum of costs of losing confidentiality of this asset with respect to any possible combination of other assets. $w(o \mid x)$ is a probability of compromising an asset o given that a considered subset of assets $x \in 2^{Asset_Y}$ has been compromised too.

To give an example of how this can be applied to confidentiality loss of a system, we consider a system Y with two assets o_1 and o_2. We substitute a simple expression for cost in equations (6.2) by a full cost function derived in equation (6.23).

$$\begin{aligned} CL(Y,t) = {} & p_{o_1}(t)cost'(o_1) + p_{o_2}(t)cost'(o_2) = \\ & p_{o_1}(t)\Big(p_{o_1|\bar{o_2}}(t)cost(o_1 \mid \bar{o_2}) + p_{o_1|o_2}(t)cost(o_1 \mid o_2)\Big) \\ & + p_{o_2}(t)\Big(p_{o_2|\bar{o_1}}(t)cost(o_2 \mid \bar{o_1}) + p_{o_2|o_1}(t)cost(o_2 \mid o_1)\Big) \end{aligned} \tag{6.24}$$

By expanding expression (6.24) and by applying the formula for a product of two dependent events, we obtain the following result:

$$CL(Y,t) = p_{o_1\bar{o_2}}(t)cost(o_1 \mid \bar{o_2}) + p_{o_2\bar{o_1}}(t)cost(o_2 \mid \bar{o_1}) + p_{o_1o_2}(t)\big(cost(o_1 \mid o_2) + cost(o_2 \mid o_1)\big) \quad (6.25)$$

According to equation (6.25) confidentiality loss for a system that has two assets is the sum of three components: (1) the loss associated with only o_1 being compromised; (2) the loss of only o_2 being compromised; and (3) the loss of both assets being compromised. Another way of looking at the notion of cost (or consequence of a compromise) can be the actual costs of recovery, i.e. the risk to the whole system with respect to its assets is equal to (1) the costs associated with recovering a system when only one of the assets is compromised (2) and the costs associated with recovering when both assets are compromised. Note that these costs may not be the same and that is what we account for.

It is important to mention that the probabilities that correspond to weights in equation (6.23) can be computed using the formal models introduced in Section 6.2. However, in cases when high precision is not required, we suggest to consider using a simple average instead of the weighted sum. For instance, the use of precisely computed probabilities for the weighted sum will be an overkill when the cost function is designed based on expert estimates that possess a high degree of uncertainty by nature.

6.5 Discussions

In this section, we discuss different insights around the proposed metrics and their derivation method gained in the course of this work.

Attack modelling is central to security research. In common with current attack-based analysis frameworks, our methodology deals with known attacks as opposed to unknown zero-day attacks. Preparing a system for zero-day attacks is also an important concern, but most embedded systems fail to protect against even known attacks due to a lack of security considerations already at the design phase and also due to general unawareness about or oversight of potential security problems [11]. Moreover, Bilge and Dumitras [135] demonstrate that the number of launches of an attack when it is transformed from a zero-day to known attack only grows. Nonetheless, predicting unknown attacks is an emerging research challenge. Todays research efforts on predicting and discovering security vulnerabilities [136, 137] is aligned with this task.

In its turn, when known attacks are considered, it is important to examine a set of attacks that are reasonable for a particular domain to obtain meaningful results. The question that arises in this context is how to obtain this set of attacks. For this purpose, SEED promotes to employ repositories designed to constantly collect all known attacks, so that a system engineer

can reuse this knowledge. To fulfil this mission, as any repository, it should be regularly updated and enriched with new information.

Among prerequisites for conducting quantitative analysis is an assumption that quantitive characteristics of an attack model and a system are determined. One can potentially argue about difficulties of obtaining realistic data about the timing aspects of an attack and system at the design phase, and therefore, question reliability of results of the proposed security analysis. We nonetheless propose that an easier and more effective exploration of security threats and impacts is already a valuable input to design decisions, even when it is subject to some uncertainties. This enables a 'what if' approach which allows understanding the sensitivity of the system to potential attacks and design decisions. For example, one can investigate which design alternative is more or less sensitive to certain attacks by looking at trends in variations of confidentiality and integrity losses. One can spot some jumps and drops in these trends that may indicate the need for further investigation. One may not be concerned about the difference between 0.2 and 0.25 (on the scale from 0 to 1) of calculated values for the metrics considering this difference be insignificant given input data uncertainties. However, one will be more careful comparing two assets that yield 0.2 and 0.9 for confidentiality loss.

The need for quantification of an attack and system models is not a new challenge. The research areas that enable quantitative estimations of timing aspects of attacks and system at earlier design stages constantly progress. Arnold et al. [131] briefly discuss two alternative approaches to obtain data about quantitive behaviour of an attack: (1) based on historical or empirical data; and (2) based on expert opinions. The most desired data, that is when the first approach is used, are actual measurements from experiments in a real environment. The main issue with this approach is the cost of such experiments. High costs can be associated with both preparation for an experiment and additional costs that can be imposed by the need to deal with possible consequences. Moreover, experiments in a real environment are often time-consuming which is also an issue in the light of time-to-market requirements coming from significant commercial pressure. As an alternative, the practitioners propose the honeypot-based technique [138, 139]. The main critique against these techniques [140] is that such experiments are conducted under controlled conditions, and therefore, can not exhibit the real behaviour of an attack. When measurements are obtained by observation or by conduction experiments (following the first approach mentioned above), one can use a range of fitting tools, e.g. G-Fit [141], and kernel density estimation tools [132] to build distributions required for our method.

As an alternative to sample-based techniques, one can employ expert-based estimates. The main advantage of this approach is that it does not require expensive and time-consuming deployment of experiments, and therefore, can be used to provide a relatively quick input. For example, Sommestad et al. report the results of employing the experts' judgements to obtain

such complex estimations as success rates of code execution [142] and denial-of-service attacks [143], effectiveness of intrusion detective systems [144], to name a few. The disadvantage will be that the use of expert estimates introduces (potentially large) uncertainties.

Uncertainties can be categorised in two types [120]: aleatory (due to inherent randomness of a considered object) or epistemic (due to lack of knowledge about an object). Epistemic uncertainty is generally considered as reducible by collecting more data and measurements; whereas aleatory is irreducible. Both these uncertainties are present when attack and system models are quantified at the design phase. Epistemic uncertainty can be addressed by refining an attack model or by considering a larger group of experts. Aleatory uncertainty is addressed in attack modelling by employing stochastic models where system and attack dynamics are assessed by assigning probability distributions. Parsons [145] extensively discusses different types of imperfect information, its source, and also methods to handle such information. Feng and Xie [146] propose an algorithm that combines two sources of information (expert knowledge and knowledge from observed cases) by constructing a Bayesian network. In our work, we use probability distributions for quantification of attack and system models in presence of uncertainties.

The next natural question that may arise is what form these distributions should have. In research literature, there are different opinions with this regard. Exponential distributions are widely used in attack modelling as a default choice. Arnold et al. [131] justify this choice by the fact that the exponential distribution has maximal entropy. This means that the exponential distribution is the most random of all distributions with a given mean. This is an appropriate consideration when expert estimates are used. In contrast, Nico et al. [147] point that the exponential distribution is not apt due to its memoryless property that does not account for ageing and learning curve of an attacker. Jurgenson and Willemson [148] justify the use of normal distributions for attack tree parameters by arguing that humans tend to think in terms of normal distribution, and therefore, it is the most natural distribution when human experts evaluate the parameters. In turn, Almasizadeh and Azgomi [140] suggest to use the mixture of uniform distributions. Finally, Madan et al. [149] argue that a variety of distributions should be used in the context of security analysis – hypo-exponential, hyper-exponential, Weibull, log-logistic, and so on – since various distributions can capture different particularities of attack behaviours. We agree with the need to be able to use diverse distributions depending on the nature of an attack, and therefore, we have adopted a flexible formalism – semi-Markov chains – that allows any type of distribution.

Our method assumes that system is annotated with such information as execution time distributions. The need for techniques to acquire such quantitive estimations already at the design phase in the domain of embedded systems is also a recognised research challenge. The model-based engineer-

ing community provides a set of techniques and methodologies for design-based (UML) performance estimations that aim to support early design-space exploration, e.g. COMPLEX [150], PUMA [151], Co-Fluent [152], and SPEU [153]. Our work uses these precursors and insights as a premise. With respect to this issue, we also believe that it is not appropriate to assume that there is no knowledge about system time characteristics at the design phase. As we discussed is Section 2.1, the pure waterfall life-cycle model is not realistic for modern engineering. This implies that a design model can be actually annotated with performance characteristics reusing data from the previous iterations. This case is in the spirit with round-trip approaches of system refinement developed in the area of the model-driven engineering [154].

7
Related Work

In this chapter we describe works related to the contributions of this thesis. We discuss those works that target similar objectives or that employ similar methodologies. Figure 7.1 depicts a diagram of the research areas involved in this thesis. These are subtopics from system engineering, security engineering, and embedded systems.

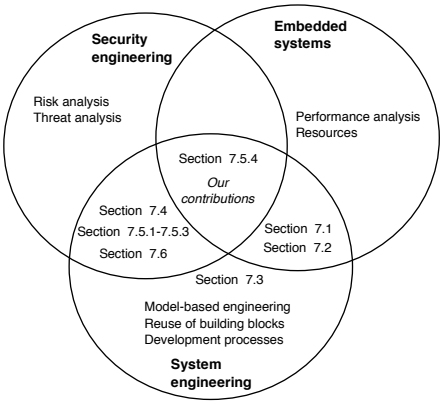


Figure 7.1: Research areas involved in this thesis

The first two sections focus on the intersection of subtopics from system engineering and embedded systems. In particular, Section 7.1 describes approaches for composing a system from reusable elements; and Section 7.2 is concerned with conduction of performance analysis of embedded systems

at the design phase.

SEED rests on combining ontology and model-based technologies to support the processes of capturing and applying security knowledge and reuse. We summary existing works that target this challenge in Section 7.3. This section belongs to the system engineering circle in Figure 7.1.

The next section covers a topic on the intersection of security and system engineering. Thus, Section 7.4 describes works concerned with modelling of security knowledge.

We dedicate Section 7.5 to methods for designing of security-enhanced systems. In this part, Sections 7.5.1 – 7.5.3 review works that are lies on the interception of security and system engineering areas in Figure 7.1; and Section 7.5.4 is related to a crossover of all three research areas.

Finally, Section 7.6 is concerned about risks and attacks. Two subsections of this part belong to the intersection of system and security engineering areas.

7.1 Composing a System from Reusable Blocks

The growing complexity of systems motivates methods for their construction as composition of reusable blocks. This approach brings a range of benefits: it allows reducing the development time and cost, improving the quality of reusable artefact, utilising better expertise of engineers from various domains, to name a few. In our work, we exploit the MBE method SPACE (reviewed in Section 2.2.4) where the notion of a building block is used. However, there are other widely spread approaches that can potentially support reusability of security solutions across different systems. In this section, we describe two such approaches, namely component-based (Section 7.1.1) and aspect-oriented paradigms (Section 7.1.2). Section 7.1.3 positions SEED with respect to these works.

7.1.1 Component-based Development

A component is a core concept used in component-based approaches for system development. It is a reusable block that encapsulates some system functions or services. The widely accepted definition of a component is given by Szyperski [155]: *"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."*

Originally, components are meant to be delivered as binary units deployed and composed at run-time. However, this requirement is often moved in case of embedded systems to the design phase due to an overhead created by a component framework [156]. We proceed to describe the basic concepts and terminology used in component-based system development approaches [157, 158].

To enable component-based design, each component must adhere a specific *component model* that defines a set of rules and conventions to describe a component. In particular, a component is defined by its *interfaces*. Each interface reflects properties of the component that are visible externally (i.e. for other components and a system as a whole). Interfaces can be represented as a set of operations with a list of input and output parameters, i.e. operation-based interfaces. Alternatively, interfaces can be considered as entities that send and receive data, i.e. port-based interfaces. One can distinguish *provided* (e.g. operations provided for their environment) and *required* (e.g. operations required from their environment) interfaces. Additionally, the notion of *rich interfaces* is introduced [157] to refer to interfaces that contain additional information about interfaces, e.g. declaration of their extra-functional properties such as the execution time. These rich interfaces enable certain verifications when composing components.

The process that establishes connection between components (i.e. the composition of their functions) is called *component composition*, *binding* [158], or *wiring* [159]. The result of composition of two or more components is referred to as an *assembly*. In some component models, the composition of components is supported through *connectors* that are mediators between components. A component model is supported (at the design- or run-time) by a *component framework* that is an infrastructure that manages resources for components.

A set of challenges arises when adopting component-based development approaches for embedded systems [160]. For example, the temporal properties of embedded systems require that components provide timing characteristics as a part of their interfaces. This is a difficult task if a component is considered to be a software unit since temporal properties depend on the underlying hardware. However, the need to utilise benefits of component frameworks for the world of embedded systems determines the tendency of developing component-based approaches for these domains as well, e.g. AUTOSAR [161] for automotive industry, ROBOCOP [162] for consumer electronics domains, and ProCom [163] for embedded systems focused on a class of such systems that perform real-time controlling tasks. Hošek et al. [164, 165] compare ten component frameworks focusing on those that provide support for execution. Besides general features related to component models and frameworks (e.g. presence of connectors, type of interfaces) the authors consider a set of requirements that are coming from the domain of embedded systems. They are, for example, support for coupling with hardware or modelling real-time attributes.

7.1.2 Aspect-oriented Development

Aspect-orientation enforces the *separation of concerns* principle [166]. This principle promotes identification of different concerns in a system and their separation encapsulating them into reusable artefacts, e.g. modules. These

artefacts can be analysed in isolation and applied in several applications. In aspect-oriented modelling (AOM), reusable modules usually realise so-called *crosscutting concerns*. The concern is called *crosscutting* if a requirement, that it expresses, cuts across a whole system. Reusable modules that implement crosscutting concerns provide such extra-functional properties as security, safety, or other quality of service properties as opposed to traditional functional units of decomposition used in component-based development. The terminology of AOM is not so mature as the terminology of component-based development. However, we have identified some basic concepts used in AOM approaches that deal with the security concern.

A *primary system model* [167] is a base model of a system (both its functionality and architecture) under development. This primary system model is further extended with a so-called *aspect* that is a reusable implementation of some function that fulfils a crosscutting concern. Each aspect has two forms: a *generic aspect* is an application independent model of an aspect, and a *context-specific aspect* is a generic aspect instantiated for a given application. To transform a generic aspect into a context-specific aspect, a set of *adaptation rules* are applied. Basically, these adaptation rules specify how to map the abstract syntax of a generic aspect into the syntax of a certain application domain. Finally, a context-specific aspect is integrated into a primary system model that results in an *integrated system model*. The integration is done by applying a set of *composition rules* that can be implemented as an engine that takes primary system model and context-specific aspects as an input and produces an integrated system model. Alternatively, composition rules can be represented as a set of instructions for an engineer that describe steps to be taken for composition.

The comprehensive and extensive conceptual reference model of aspect-oriented modelling constituents is presented by Wimmer et al. [168]. Below, we overview two AOM approaches that deal with the security and dependability concerns. These approaches employ UML for functional modelling of a system and its aspects.

France et al. [167] represent a generic aspect (referred to as "generalised form of a solution") as a pattern that describes common characteristics of a solution. These patterns are realised as UML model templates. The adaptation is implemented as instantiation of a pattern by binding (i.e. relating) its template parameters to application-specific values. The composition is realised by merging UML models of a primary system model with context-specific aspects. This merge can be controlled and managed by so called *composition directives*. For example, composition directives can be used to specify that some elements of a primary system model should be removed, added, or modified in a certain way. They also can predefine the order when several aspects are composed with one primary system model. Thus, varying composition directives several (potentially different) integrated system models can be obtained. Further, an integrated system model is analysed to reveal conflicts or undesirable properties, and to investigate whether the

used aspect provides the required level of dependability.

Mouheb et al. [169] present another approach for AOM. A primary system model is a UML model where some elements are annotated with stereotypes and tags that express security requirements, e.g. confidentiality and integrity of data. Additionally, a primary system model is extended with specifications of *join points* used to support composition of a primary system model and aspects. In particular, join points specify where an aspect should be integrated in a primary system model, e.g. before or after a specific operation. An aspect is a UML class model that is extended with a set of stereotypes from a specific UML profile provided by the authors. For example, one of the stereotypes provided by this profile is a *pointcut* that is used to specify where in a primary model an aspect should be inserted. Similar to join points, a pointcut can specify, for instance, that an aspect should be inserted before or after a UML operation or call. Each aspect can provide several functions that are referred to as *advice* in the mentioned profile. The adaptation of a generic aspect, i.e. creation of a context-specific aspect, is done as matching join points from a primary system model and pointcuts from a generic aspect. The composition is defined as actual *weaving* of aspects into a primary system model.

Aspect-oriented modelling has already been applied to development of embedded systems. For example, Wehrmeister and Berkenbrock [170, 171] use aspects, namely aspects crosscutting overview diagram, for modelling of non-functional requirements of real-time embedded systems within the AMoDE-RT (Aspect-oriented Model-Driven Engineering for Real-Time systems) design approach. The authors show [171] that one can increase the reuse of developed artefacts by encapsulation of non-functional requirements in aspects. Zhang [172] extends AADL (Architecture Analysis and Design Language) components with a set of aspects from the railway domain.

7.1.3 SEED and Reusable Blocks

Approaches that allow composing a system from reusable elements have been increasingly popular in recent research. Each approach has its own syntax and framework (i.e. theoretical foundations supported by tools) to integrate a reusable unit (i.e. a component, aspect, or building block) into a system model. When such approaches are adopted, a security countermeasure that enforces security properties can be represented in the form of a reusable unit.

We have described the two approaches in Sections 7.1.1 and 7.1.2 where a security countermeasure can be encapsulated into a component or into an aspect. In our work, we employ the notion of reusable building blocks used in the MBE method SPACE (reviewed in Section 2.2.4). By employing SPACE we enjoy a range of capabilities provided by this method. They are, for example, a tool-set based on Eclipse Modelling Framework called Arctis [42] and a rich library of already implemented RBBs [43] including the one that

model security countermeasures. Moreover, the modelling language used by SPACE allows us to analyse system models and elicit new information required to support integration of a relevant set of security countermeasures.

Our contribution enhances the step *composition and analysis* from Figure 2.9 when it comes to decide on a set of security countermeasures expressed as reusable building blocks. In particular, we elaborate methods to select a set of security building blocks that are suitable for a system under development according to identified security needs.

Notice that the ideas of the SEED approach are not limited to SPACE reusable building blocks. The SEED foundation level imposes the requirement that it should be possible to encapsulate a security solution as a reusable element. Both components and aspects satisfy this requirement. Thus, any of the above mentioned concepts (i.e. components or aspects) can be potentially employed for implementing the SEED realisation level.

7.2 Performance Analysis at Design Phase

This section reviews methods for obtaining performance estimates from design models (Section 7.2.1) and other methods that use performance estimations obtained from other sources in system models (Section 7.2.2).

7.2.1 Obtaining Estimates from System Models

There is a range of tools that enable performance analysis when only a design of a system is available. Robert and Perrier [152] and Piel et al. [173] present CoFluent and Gaspard2 methodologies and tools to execute performance analysis at the design phase. These techniques use UML/MARTE models to describe architecture and application designs. In addition to the MARTE profile, CoFluent employs the SysML [174] modelling language. Both CoFluent and Gaspard2 methodologies use a different chain of transformations to generate SystemC TLM [1] code for its further simulation in suitable tools.

The use of performance analysis while composing a system with RBBs at the early design phases is a subject of active research. Woodside et al. [175, 151] develop and apply the PUMA [2] approach for performance analysis of RBBs represented as security aspects. Woodside et al. start exploiting the UML SPT [3] profile, but their further works adapt this methodology for the MARTE profile. In this work, the authors generate LQN [4] models that are analysed by a solver and simulator. Similarly, Wehrmeister et al. [176] presents the AMoERT [5] methodology when the aspect-oriented

[1] Transaction Level Model
[2] Performance by Unified Model Analysis
[3] Schedulability, Performance and Time
[4] Layered Queueing Networks
[5] Aspect-oriented Model-Driven Engineering for Real-Time systems

paradigm is used. In this work, the authors propose the GenErTiCa tool to generate Java code for a specific predefined (though selected by an engineer) target platform. Bondarev et al. [177] present the CARAT [6] toolkit for performance evaluation where RBBs conform to a specific component model. The authors use their own modelling language to describe the application logic and architecture that are synthesised into an executable system model used for the task scheduling.

The field of design-space exploration is also concerned with estimating performance at early design phases. Herrera et al. [150] propose a methodology called COMPLEX that uses MARTE modelling. Within this methodology a simulation infrastructure (called SCoPE+) is developed that generates executable models of the system. SCoPE+ consumes a set of inputs for this purpose. Among these inputs are descriptions of hardware and allocation of functional components on this hardware. Oliveira et al. [153] present the SPEU methodology for exploration using analytical estimations.

We can distinguish two categories of methods that allow obtaining performance evaluation results when only a design model is available. The methods of the first group (CoFluent and AMoERT) take a set of application and platform models as an input and generate code, e.g. C and SystemC. Afterwards, the generated code is executed in a simulation tool. The methods of the second group (PUMA, CARAT, and Gaspard2) use models as a means to input required data into analytical performance analysis tools.

In this thesis, we capitalise on the outlined methods. The SPACE method employed in our work enables code generation for functional models (i.e. UML activities) where Java code is produced from state machines for a certain predefined execution platform (e.g. ServiceFrame [44]). Thus, one can relate the SPACE methodology to the first category of the classification outlined above except the fact that transformations are bound to a specific predefined execution platform. In our work, we complement SPACE models with an execution platform described as MARTE models. These MARTE models can be converted into the corresponding simulation code using methods developed in the approaches mentioned above. Therefore, there is a potential to simulate SPACE models on a desired execution platform (that is modelled in MARTE), once a formalised and tooled allocation of SPACE models onto MARTE models is defined. Moreover, our work complements the approaches described above since it enables reuse of outcomes of SBBs performance evaluation conducted by corresponding domain experts.

Note that the outlined above approaches (e.g. CoFluent and Gaspard2) introduce some constraints on the original semantics and on usage conventions of MARTE models to enable their further analysis. Similarly, we make some assumptions on MARTE models when designing the support (methods and tools) for model-based compatibility analysis.

[6] Component Architectures Analysis Tool

7.2.2 Using Estimates in System Models

Ciccozzi et al. [154] propose a meta-model used to propagate results of monitoring extra-functional properties at the code level back to a system model in order to improve it. The authors refer to this approach as round-trip support and the meta-model is called as back-propagation meta-model. This approach is developed for the CHESS modelling language [178].

We find that our approach shares the idea of using values calculated at the later development phases (code level) for refinement of a system model. Thus, we promote to use information about performance of SBBs (possibly obtained when experimenting with already implemented artefact) for making decision at the design phase. In our work, we exploit MARTE models to capture platform-specific constraints of embedded systems and SBBs. The proposed MARTE compatible profile allows capturing more information about SBBs performance analysis. This includes the outcomes (also captured by Ciccozzi et al.) as well as the used workload and the observed resource footprint. Elaboration of a back-propagation model goes beyond our scope, but we find that it can greatly facilitate the task of feeding information into our performance evaluation profile and ontology. This information (structured, captured, and searchable) aids an embedded system engineer to select an appropriate SBB to satisfy required security (or other extra-functional) properties.

Desnitsky et al. [179] present an approach (referred to as configuration model by the authors) to find an optimal set of security components (similar to SBBs) based on information about embedded system capabilities (available resource) and security components demands. This data is manually entered into a tool. Configuration is implemented as multi-criteria optimisation problem on a set of security components.

In SEED we develop a technique for compatibility-based selection of SBBs. We find the configuration technique proposed by Desnitsky et al. [179] as complimentary to our model-based compatibility analysis method that can be realised on top of the developed resource ontologies. In continuity, Desnitsky and Kotenko [180] also propose to take into consideration hidden conflicts when selecting security components. The author distinct three types of conflicts: type 1– conflict due to a lack of consistency between a security component and the device specification; type 2 – conflict between the protection functions of several security components; and type 3 – conflict between several basic components within a complex security component. In this vein, our compatibility analysis can be seen as a technique supporting identification of conflicts of type 1. Identification of conflicts of type 2 can be supported by our security ontologies given that the conflicting relation is added directly into the ontologies or other rules for identification of such conflicts based on knowledge stored in the ontologies are elaborated.

7.3 Marrying Ontologies and Models

The use of ontologies to support tasks of model-driven engineering is an interesting research topic [32]. The potential of ontology technologies applied to the system and software engineering to formalise general modelling is outlined by Tetlow et al. [181]. Recall that ontologies are used to represent knowledge as a set of domain concepts and their relations. Similarly, the conceptual description of a domain through meta-modelling is performed while creation of a DMSL [36]. Both technologies suggest a range of benefits. For example, one of the main benefits of the ontology technology is its automated querying services while DSMLs enjoy wider adoption in development environments and tools. Therefore, it is not a surprise that researchers try to find a logical synergy to exploit advantages of both of these technologies to facilitate designing of complex systems.

Walter et al. [182] in their recent work employ ontologies to improve the practice of domain-specific modelling (DSM). The authors have developed a framework for DSMLs that relies on the ontology reasoning services (e.g. the inconsistency checker) to guide a designer and to validate incomplete structural domain models. Dibowski et al. [102] present the ontological framework to describe devices for the building automation domain. Jianjun et al. [183] use ontology to configure embedded control systems based on a functional model of a system that is intended to express the user demand. Wagelaar [184] combines the ontology technology with the model-driven architecture principles to enable reuse of platform-independent to platform-specific models (PSMs) transformations. Tekinerdoğan et al. [185] employ ontologies to support selection of PSMs, where a system platform is described as a set of high level properties.

Even thought inspired by the same idea of combining ontology and modelling technologies, we employ this synergy for a quite distinct purpose and in a different way. In our work, we combine the ontology and DSM technologies to assist the development of security-enhanced systems. First, we employ DSM to constantly populate the developed ontologies with the domain-specific security knowledge. Similar, to the works mentioned above we use knowledge stored in the ontologies to guide a system engineer, but our intention is selection of SBBs. Furthermore, we use MARTE models and extend them with additional concepts to formulate platform-specific constraints as a basics for composition of a system and SBBs. Using these constraints, we elaborate on the notion of model-based compatibility as one of possible criteria for selection of a set of SBBs.

7.4 Modelling Security Knowledge

To support knowledge intensive tasks in a semi-automatic way a structured representation of this knowledge is required. Ontology is a suitable technique for this purpose. A number of ontologies for capturing security knowledge

have been proposed.

Herzog et al. [72] and Fenz and Ekelhart [73] introduce two ontologies that formalise the domain of information security from different aspects; Kim et al. [74] present an ontology for annotating web-services; Karyda et al. [75] propose an ontology to assist reuse of the experts' security knowledge in the area e-government applications. Extended surveys and classification of security ontologies can be found in works of Blanco et al. [186], Souag et al. [187], and Gärtner et al. [188] where the authors discuss 28, 17, and 14 security ontologies respectively. In total, we can see 41 distinct security ontologies proposed by researchers. In our work, we adopt the ontology presented by Herzog et al. [72] since it is built upon classic components of risk analysis.

Souag et al. [189] try to build a complete ontology to support security requirements elicitation based on their earlier survey [187]. The authors evaluate on 10 experts their hypothesis whether a complete and usable ontology will suffice to support security requirements elicitation. The results show that only a good ontology is not enough. Souag et al. conclude that a bridge between requirements elicitation and security knowledge (ontologies) is needed. With respect to this observation of the authors, we equipped a system engineer with the set of methods that provide a bridge between the security knowledge stored in the ontologies and the system design models.

Once security knowledge is acquired it is important to maintain its freshness and consistency. It is also important to establish a strategy for knowledge integration and sharing among different projects. These concerns have been left outside the scope of this thesis. However, these aspects are addressed by other researchers.

For example, Ruhroth et al. [190] discuss this topic and propose a workflow for ontology adaptation. The author extend the standard OWL's import mechanism with new operators (e.g. add, hide, change) that can be used to adapt ontologies for project-specific needs. These new operations can be used, for example, to hide a part of and ontology if it is needed. To support consistency of an ontology while updating it, Javed et al. [191] propose to follow a set of change patterns (e.g. for concept splitting and merging). These patterns are also adopted by Ruhroth et al. [190].

7.5 Security-enhanced System Design

This section summarises our review of methods developed to assist a system engineer in integrating security mechanisms into a system design. We start discussing methods to deal with security aspects that are designed for general systems in Section 7.5.1 followed by ontology-based approaches in Section 7.5.2. Thereafter, Section 7.5.3 briefly describes approaches related to selection of security measures given a repository of available alternatives. Finally, we discuss methods that target embedded systems in Section 7.5.4.

7.5.1 General Methods to Deal with Security

The challenge of integrating security mechanisms into various types of systems has been addressed by several approaches. They are, for instance, MDSE[7]/UMLsec, MDS[8]/SecureUML, security aspects, security patterns, and AVATAR, to name a few.

MDSE/UMLsec [109, 192, 193] is one of the first approaches developed for integration of security related information into UML specifications of a system. In particular, UMLsec is a UML profile that is used to incorporate security requirements (such as the fair exchange principle and secure communication links) in the form of corresponding stereotypes (i.e. *"fair exchange"* and *"secure link"* respectively) and their tags (e.g. those tags that allow specifying an adversary) into various UML models (e.g. class or activity models). These stereotypes allow a system engineer to specify a proper set of security requirements for a system under development. Thereafter, a system design should be enhanced to meet the augmented security requirements using such techniques as, for example, security patterns [108] or direct refinement of an initial system design. The use of UMLsec allows formally verifying if the resulted system design meets the annotated security requirements. Additionally, there are some works that complement the UMLsec methodology with the security requirements elicitation phase. Thus, Houmb et al. [194] propose a methodology to elicit requirements from stakeholders employing a heuristics-based tool empowered by the common criteria standard [195].

MDS/SecureUML [196, 197] deals with design and verification of role-based access control systems. SecureUML is a security modelling language that should be combined with other languages used for a system design. Thus, it enables a system engineer to model both security and functional aspects of a system simultaneously. The merge of languages is proposed to be implemented through a so-called "dialect". A dialect is used to match meta-models and syntax of corresponding functional and security languages. In particular, a dialect shows what elements of a system design language are SecureUML resources and what actions are applied to these resources. Thus, SecureUML can help to express such information as which elements of a system model shall be considered as resources and what actions are permitted under these resources.

The aspect-oriented paradigm presented by Georg et al. [198] identifies security as a crosscutting concern. To deal with this concern, security functions are encapsulated as aspects that are woven into the initial (primary) system model. In this approach, security requirements are elicited by means of composing a system model with a threat model and checking if the attack succeeds. Thereafter, a suitable security aspect is encapsulated into a system design. Aspect-Oriented Risk-Driven Development (AORDD) is an

[7]Model-Driven Security Engineering
[8]Model-Driven Security

approach developed by Georg et al. [238] (that extends the earlier contributions [198]) to calculate the fitness of different security countermeasures as a trade-off between different criteria, e.g. provided security level, project and deployment effort. The fitness score for a particular security countermeasure is calculated with a specially constructed Bayesian Belief Network. Further, Houmb et al. [199] add to AORDD the performance analysis step employing the PUMA tool.

Security patterns [108, 200] are intended to capture security solutions for common security challenges. In general, a pattern is an extremely broad concept that can be used to encapsulate expert knowledge of any kind (e.g. a reoccurring structure, process, activity, or just some kind of "thing") along all phases of a system development. To be able to cover such diverse information, patterns are defined in a highly generic form. In particular, each pattern describes a solution in a human-readable text (sometimes referred to as patterns's documentation), which is sometimes augmented with UML models to help developers to understand a pattern. Schmidt et al. [201] develop an approach called Security Engineering Process using Patterns (SEPP). In this process, the authors introduce a special type of patterns called Security Problem Frames (SPFs). SPFs consider generic security requirements omitting possible means to satisfy these requirements. The SPF description contains the following fields: name, intent, a frame diagram, informal description, a security template, and an effect. Each field is defined in a textual form, as a UML model, or in some other formal notation. Thereafter, the authors introduce the notion of a Concretised Security Problem Frame (CSPF) that describes generic security countermeasures linked to related SPFs. It is the responsibility of a system engineer to select and instantiate both SPFs and CSPFs to proceed with the SEPP method. Uzunov et al. [202] survey the state-of-the-art in security patterns and methodologies for applying them for securing distributed systems.

Ramón et al. [203] proposes an approach for automatically generating security software artefacts from security models called ModelSec. The authors propose to separate modelling of system requirements from modelling of security requirements. The authors build a dedicated DSML for modelling of security requirements. When security requirements are modelled, abstract security mechanisms (called security design models) are mapped to corresponding security requirements. This mapping is built of two parts. First, a part of the mapping is captured in the model-to-model transformation (security requirements to a security design model). For example, authentication requirements are mapped to *AuthenticationDecision*. Thereafter, a skeleton obtained after applying the transformation must be completed by engineers with information related to the design of the system. The system design information is still abstract in the security design model (e.g. EJB). When a security design model is constructed, a security implementation model that expresses the implementation details of a concrete target platform is built around this security design model (e.g. BEA WebLogic

that is an implementation of EJB).

Other approaches that are designed to assist a system engineer to secure a system can also be mentioned. For example, Hamid et al. [204] enforce the notion of security (and dependability) patterns supported by formal validations. Pedroza et al. [205] propose a SysML-based environment called AVATAR to model and verify safety, authenticity, and confidentiality properties. Sectet [206] deals with integration of security requirements into inter-organisational workflows by handling security policies.

The recent work of Uzunov et al. [207] provides a comprehensive survey, comparison, and classification of many other methodologies to assist in designing security-enhanced systems. Other surveys and systematic reviews are conducted by Basin et al. [197], Kasal et al. [208], Nguyen et al. [209], Jensen and Jaatun [210], and Lucio et al. [211].

Compared with the SEED approach proposed in our work where the security-related knowledge is captured by DSSMs, the approaches mentioned above still require in-depth security knowledge from a system engineer or presence of a security expert. For example, it is not clear how a system engineer should select a suitable security aspect in the AORDD method. It is assumed as the given knowledge since the authors know a priori how to model relevant attacks and how to construct and weave a security aspect. In the SEPP method, selection and instantiation of both SFPs and CSPFs are supposed to be done by a system engineer that can be cumbersome due to their complicated structure. In contrast, we provide a bridge between security domain experts and embedded system engineers. The SEED approach defines a structured methodology (i.e. concepts, methods, processes, and tools) to create and select a suitable set of security measures represented in the form of reusable building blocks that can be integrated into a system under development. Thus, our approach produces a guideline on where and why a specific security countermeasure should be applied.

A security pattern is a powerful concept to assist in analysing and developing secure software in a systematic manner exploiting the "body of accumulated knowledge" represented as a pattern. However, due to its unfixed (though flexible) syntax, their application is thought to be manual. This statement is supported by the fact that security patterns, in general, do not make any assumption on the used design language for a system implementation. Thus, application of current security patterns can be hardly automated.

ModelSec shares our intention to collect security concerns in a separate model (called security requirements model in ModelSec), but the authors do not elaborate further on the idea of separating security experts and system engineers constantly mixing these two roles. In contrast, SEED allows separating the task of designing a security mechanism from a system design by exploiting the principle of application domain specialisation.

Once a security mechanism is integrated into a system model it is important to verify that it is properly integrated and provides claimed security

properties. SEED explicitly prescribe this step (see step 7.a in Figure 5.17). Many of the methods mentioned above provide some means for conducting such security analysis. UMLsec, for example, complies an integrated model into a set of first-order logic axioms and verifies them by a prolog-based tool. SecureUML, in turn, uses the SecureMOVA tool that allows verifying security properties expresses as OCL constraints [212]. In the presented version of the SEED realisation we enjoy verification utilities provided by the SPACE method (see Section 2.2.4 for more detail).

7.5.2 Ontology-based Approaches

There is a set of works that share our inspiration of using ontologies for capturing security knowledge and applying it afterwards. These are, for example, works of Gärtner et al. [188] and Daramola et al. [213]. The authors focus on security requirements, which are use cases written in a natural language, while we work with design models that represent system functionality and execution platform. Both works use misuse cases for security analysis, but when Gärtner et al. [188] generate them based on security-domain knowledge applied to regular requirements use cases, the approach suggested by Daramola et al. [213] requires that such misuse cases are formulated by a system engineer as an input into the elicitation process. To generate misuse cases Gärtner et al. [188] use heuristics (captured by the ontologies) to identify parts of requirements that are susceptible to security issues. In our approach we have developed a set of methods for this purpose. However, we think that these methods can be only enriched by using other heuristics applied to design models.

Kang and Liang [214] propose an ontology-based method for MDA (Model-Driven Architecture). For this purpose, the authors develop a set of new ontologies: ontologies to support security analysis at the PIM (Platform Independent Model) level; ontologies to support security analysis at the PSM (Platform Specific Model) level; and ontologies to support security analysis at the code level. In these ontologies the authors capture only relation between concepts located in different levels of abstraction (i.e. PIM, PSM, code), while within one level of abstraction concepts are not interrelated. For example, it is clear that attack should be related to assets and risks, and security goals (confidentiality) must be related to assets. By omitting these details, the authors lose a lot of relevant information. Moreover, according to the authors to identify security issues a system engineer needs to be aware about security (that is often not the case) or try to identify such issues by studying the ontologies. Similarly, we have developed a set of ontologies (core security ontology, core evaluation ontology, etc.), however, our intention is to use these ontology in methods (e.g. asset identification) to support a system engineer by (semi-)automatically identifying security issues in a system model. Finally, we also provide a process for security experts to constantly enrich the ontology, while Kang and Liang [214] assume

that the ontology will not evolve.

Dritsas et al. [215] develop an ontology with an intention to support design and development of e-commerce systems. However, the support is limited to asking a proper competence questions formulated by a system engineer. While our ontology can also be adapted for this purpose (as any ontology) we also provide extra support in a form of three methods described in earlier chapters.

7.5.3 Selection of Security Countermeasures

An important step that precedes actual integration of security building blocks is the selection of these. We have observed that the model-based (model-driven) methods mentioned above do not address this task in detail. However, approaches for selection of security measures are under development in the context of security patterns [108].

Selection of security patterns is based on their classification. Such classification can be built of just three classes (Fernandez et al. [216]) or be represented as a multi-dimensional matrix of classes (VanHilst et al. [217]) including such categories as an architectural layer (e.g. network), a lifecycle stage (e.g. design), and a domain (e.g. enterprise systems). In our work the basics for selection of concrete SBBs are domains, security goals, defence strategies, and assets. Thus, there are some overlapping concepts compared to the dimensions elaborated for security patterns. However, SEED proposes that only the domain dimension is selected by an embedded system engineer following the guideline. Selection of other categories is done by applying the elicitation technique and querying the security ontologies.

Hafiz et al. [218] organise security patterns according to the CIA[9] goal model, a pattern's application context (i.e. core security, perimeter security, and exterior security), a problem domain, and their classification based on the STRIDE[10] model. All these classifications are formalised in tabular or tree forms. In addition to this method, Washizaki et al. [219] develop the *Dimensional Graph* (DG) concept to formalise the multi-dimension classification of security patterns. Each DG uses a UML object diagram to show relations of a pattern to a set of dimensions of interest. We formalise our categories as the core security ontology, which allows utilising advantages of the ontology technology querying services for the selection of concrete SBBs.

Nguyen [220] propose to use a feature model from the software product lines domain. This feature model describes in what ways security patterns can differ from each other. The authors distinct five types of relations between patterns: "depends on", "benefits from", "alternative to", "impairs", and "conflicts with". In the current state of our core security ontology, two out of these five relations are represented. These are "depends on" (when

[9]Confidentiality, Integrity, and Availability

[10]Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege

a SBB issues another asset that also requires protection) and "alternative to" (two SBBs are alternative when they provide similar security properties). We believe that the core security ontology will only benefit when it is complemented by additional relations between SBBs as discussed by Nguyen [220].

7.5.4 Methods to Deal with Security for Embedded Systems

A number of model-based approaches to enforce security within embedded systems are proposed. The Ruiz et al. [221, 222] approach requires that a system engineer defines a set of security properties for a scenario. This assumption already requires that an embedded system engineer has a good expertise in security. Each security property is further linked to threats and attacks through so-called threat model. Attacks and threats are modelled in order to describe the capabilities of intruders to cause harm in a system. Finally, these threat models are associated with a set of static and dynamic tests to enable the later testing of a system integrated with security mechanisms. In our approach, we propose a method to systematically elicit required security properties consulting the security knowledge captured in DSSMs. Therefore, our approach addresses the situation when an embedded system engineer has just limited expertise in security. We believe that our approach can be complemented by the Ruiz et al. [221] idea to link threats with a corresponding set of tests. However, we envisage that there is also a need to extend this idea linking security properties and threats to verification facilities (besides tests). This enhancement will provide required assurance already at the design phase as opposed to postponing all checks until the first prototype of a system is available when testing can be performed [223].

Hamid et al. [224] attempt to model trust properties as reusable patterns for specific domains. While that work shares our aspirations for reusability, we consider security concerns rather than trust relations.

Similar to our work Eby et al. [225] adopt principles of domain-specific modelling. They propose to integrate a Security Analysis Language (SAL) into a DSML for the embedded systems domain (that reminds us the SecureUML approach). However, they focus on security of information flows.

Saadatmand and Leveque [226] develop a method for incorporating security aspects into the ProCom component model. The authors consider two security goals, namely confidentiality and authentication. This approach proposes to use (manual) annotations to identify those parts of a system model where integration of security aspects is needed. In contrast to this work we have developed the asset elicitation technique to identify vulnerable parts of a system avoiding manual tagging of a system model.

Apvrille and Roudier [227, 228] propose a model-driven environment for developing secure embedded systems called SysML-Sec. The authors pro-

vide a set of extra constructs (based on SysML) for adding security related information into a system design as well as connect their environemnt to some verification facilities, e.g. for evaluation of real-time constraints in presence of integrated security mechanisms. The main intention of the authors is to support collaboration of system designers with security experts; whereas SEED aims at providing a means to bring security expertise to system engineers when security experts are not easily available within the development process.

Fayyad and Noll [229] report the recent development of the nSHIELD project (new SHIELD). This project is an extension of the earlier SHIELD project. In combination these two research projects address a complex problem of analysing security, privacy, and dependability (SPD) attributes of embedded components and a system of interconnected components. Obviously, the scope of this work (and these two research projects) overlays with SEED, but it is also clear that the scope of nSHIELD goes beyond our focus. Thus, the authors target to provide an aggregated metric that accounts for the SPD triple (a SPD level metric), while we focus on only the security attribute. The authors look at the large context of system-of-systems, while we focus on supporting system engineers in designing security-enhanced embedded systems. We think that extension of the scope of our work to the system-of-systems level is a valuable direction for SEED.

7.6 Risks and Attacks

In this section we focus on a group of methods that support quantitative analysis of system security. In particular, we look at risk and attack modelling methods and mainly relate them to the method for calculating CL/IL metrics presented in Chapter 6.

7.6.1 Risk Analysis

There are general methods that specify main steps of any risk analysis. These methods are CRAMM (CCTA Risk Analysis and Management Method) [230], ISRAM (Information Security Risk Analysis Method) [231], OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation) [232], Boyer and McQueen [233], etc. These methods usually prescribe basic steps of risk analysis while leaving out details of their implementation.

A set of standards addresses security risk management, for example, NIST SP800-30 (Risk Management Guide for Information Technology Systems) [234] and ISO 31010 (Risk management – Risk assessment techniques) [235]. NIST creates foundations of risk management program and contains basic guidance that broadly covers conduction of risk assessment of federal information systems and organisations. As pointed by Lund et al. [106] this guidance should be complemented with a risk analysis process and cannot be considered as full fledged analysis method. ISO 31010 is a

supporting standard for ISO 31000. Similar to NIST SP800-30, ISO 31010 provides guidance for conduction of risk assessment, but does not specify any type of method or techniques that can be used in a specific application domain. CORAS is based on ISO 31000. Our work can be considered as contributing into a range of risk assessment techniques that are specific for design phases of the embedded system development.

Several models for risk evaluation exist that have slightly different ingredients. For example, risk for physical security is often modelled as $R = P \times V \times D$ [236, 237] where P is the probability of threat occurrence, V is vulnerabilities of the system, and D is consequences. Another model represents security risks as $R = U \times C$ [106, 234] where U is the likelihood of unwanted incidents and C is consequences. These models resemble each other, i.e. the unwanted incident component in the latter model combines the first two components in the former. We base our work on the second model and focus on evaluating the U component.

Next relevant area of research is model-based risk analysis methods which encompasses such methods as CORAS [123], AORDD [238], and CySeMoL [239]. While following the same basic structure of risk analysis, these methods provide richer support for analysts.

CORAS [106] is a general approach to risk analysis (well-established and already applied in numerous domains). It specifies 8 steps. CORAS uses a graphical notation to structure the elicited assets, threat scenarios, related vulnerabilities, and unwanted incidents. This notation is supported by formal calculus that enables calculation of risks. CORAS has a broad scope and can be adopted for a large variety of systems (not even limited to IT systems) and on any stage of a system development life-cycle.

We aim at providing tools for system engineers when designing a system. In particular, the output provided by our method can be used by system engineers to reason about the protection needed for data assets in the context of a given design. The main input of our method is a system model that is further formally analysed to obtain a risk value. System modelling (target of the analysis) is also strongly encouraged by CORAS, however, the main source of inputs in CORAS is a serious of workshops and interviews where modelling is used to facilitate communication between the analyst team and stakeholders. In our method there is a set of inputs that can be obtained as a result of such workshops, e.g. attack step probability distributions and cost functions for assets. It goes beyond our purposes to define a rigid structure of these workshops. Here, we can envisage that the principles for this process defined by CORAS give strong foundations. Similar to CORAS, our approach can be classified as asset-driven risk analysis method since it focuses on evaluating risks with respect to data assets present in a system under consideration that are identified early in the risk analysis process. However, we focus on data objects as assets while CORAS can work with any notion of assets conditioned by the fact that CORAS is developed to work with a large variety of systems. Our scope is narrower (embedded systems)

that, in turn, allows us to provide additional assistance for system engineers when identifying the assets (e.g. the asset elicitation technique). Finally, it is worth mentioning that CORAS also supports such important tasks as documentation, facilitating communication during structured brainstorming sessions, giving advice on changes, and has additional support to deal with legal aspects. These constituents are outside of the scope of this thesis.

Sommestad et al. [239] propose a method for risk assessment of enterprise systems. It has also been applied for SCADA systems. The authors' intention is to connect system architecture models to cyber security assessment concepts. The employed theory is probabilistic rational model (a version of Bayesian networks). In this framework the authors define a 3-layer of abstraction. At the first layer there is a model of the security domain (one permanent model) called an AbstractPRM created based on Common Criteria [195]. Besides entities and relations it contains also quantifiable conditional dependencies, which values are refined on the next two layers. The second layer is a ConcretePRM that specialises subclasses and relations from the AbstractPRM according to a system domain, e.g. asset is refined as DataStore, Host, and Zone (the same is done for countermeasure, attack, etc.). Finally, the third layer is an instance of the ConcretePRM. This method is implemented within the framework called CySeMoL (Cyber Security Modeling Language). Thus, the main idea of CySeMoL is to capture dependencies among different elements of risk analysis and system architecture, so that a system engineer is equipped to derive risks associated with a certain architecture.

Similar to our work, CySeMoL is intended to relax the need of having a security expert closely involved in the loop when analysing security of a system. This distinguishes CySeMoL and our approach from the CORAS methodology where security experts are closely involved into the risk analysis process. On one hand, this enables creating additional support; however, on the other hand, it limits the scope of these methods. In the CySeMoL framework, all information related to security is implanted into the ConcretePRM; similar, in SEED the information related to assets, attacks, and countermeasures is fetched from the SEED repositories. With CySeMoL the authors aim to provide a means to analysis "system-of-systems" security. By defining confidentiality and integrity losses metrics, we focus on providing a means for security analysis of design models of embedded systems that are usually relatively small systems. Obviously, it is impossible to guarantee security at the system-of-systems level when there are no considerations for security at its ends that are often embedded systems in the modern world. Thus, our method can be considered as complementary to CySeMoL.

Aime et al. [240] aim to identify suitable sources of data to conduct risk analysis and to formalise it so that this data can be utilised for automatic elaboration. For this purpose, the authors develop an experimental framework (that targets mostly enterprise IT systems) called AMBRA. For processing, one needs to provide a system description in special languages:

Positif System Description Language (PSDL) that allows describing software and hardware infrastructure; W3C's Web Service Choreography Description Language (WSCDL) to describe provided by a system services; and Positif Security Policy Language (PSPL) to provide security policy descriptions. The main mechanics for identification of known vulnerabilities in a system design is pattern matching. The authors slightly discuss potential security metrics. In particular, they focus on different combinations of vulnerabilities present in a system model, e.g. counting vulnerability or summing up their scores.

Similar to Aime et al. [240] we rely on formal descriptions of a system for automating asset identification. However, in our work we derive these formal descriptions from conventional engineering artefacts. The main benefit of such an approach is that an existing models created in a course of designing a system can be reused for security analysis. As opposed to counting vulnerabilities, we provide probabilistic risk metrics that accounts for uncertainties naturally present due to unpredictable behaviour of attackers.

Surveys of risk analysis methods and corresponding metrics are presented by Verendel [241], Sulaman et al. [242], Rudolph and Schwarz [243], Krautsevich et al. [244], and Jansen [245] among others. For example, Verendel [241] analyses more than 100 approaches and metrics to quantify security. The conclusion of the author, that is highly relevant for our work, is that CIA (Confidentiality, Integrity, Availability) quantification methods are underrepresented. Sulaman et al. [242] study 57 papers. The authors conclude that most of the existing methods are qualitative and not quantitative. We contribute to the class of quantitative methods. Sulaman et al. also state that most of the methods are developed for IT systems in general and not for specific types of IT systems. With respect to this observation, we provide support for the design phase of embedded systems.

Chen et al. [246] present a holistic approach for assessment of cybersecurity of complex computing systems. The authors point out the importance of using workflows (descriptions of how a system provided its functionality) as basics of cybersecurity analysis. Chen et al. use a wide range of sources of information for their analysis. These are security goals (e.g. availability), workflow description, system description (e.g. network topology, configuration of devices and subsystems), attack model (e.g. attacker strategies and entry points), and evidence (e.g. probability to success). These information is used to construct three types of graphs. These are G-graph (that combines security goals and workflows), GS-graphs (that embeds system descriptions into a G-graph), and GSA-graph (that embeds attack models into a GS-graph). These structures (generally referred to as security argument graphs) are generated automatically (details are presented by Tippenhauer [247]). Vu et al. [248] describe a tool called CyberSAGE that support the presented approach.

We find that Chen et al. approach differ from SEED from many perspectives and at the same time it shares a lot of commonalities with SEED.

For example, SEED emphasises knowledge capturing and reuse of it, while this concept is not distinct in the Chen et al. approach; for quantitative analysis we employ Markovian processes, while Chen et al. exploit Boolean algebra for the assessment; we focus on certain modelling languages (e.g. MARTE, SPACE), while Chen et al. leave these details outside of their approach. As we mentioned above, Chen et al. [246] use a wide range of sources of information for their analysis. One can draw similarities to information consumed by our methods, i.e. security properties (comparable to security goals of Chen et al.), functional models (comparable to workflow descriptions of Chen et al.), execution platform models (comparable to system descriptions of Chen et al.), attack models (comparable to attack models of Chen et al. annotated with quantitative information, i.e. evidence in terms of Chen et al.). Differently from our methods, Chen et al. generate intermediate graphs (i.e. security argument graphs) for conducting analysis. One of the benefits from generating these intermediate structures is that they can be used for understanding the causes of obtained results. This, for example, can be a valuable input for debugging. We believe provision of such additional information will only strength SEED.

Being a complex subject, security can be measured from different perspectives. Sanders [12] provides classification of security metrics into organisational, technical, and operational security metrics. Organisational metrics describe how effectively organisational programs attain cybersecurity; technical metrics score the system security level; and operational metrics evaluate risks to an operational environment of a system. The authors also emphasises that security metrics should be used to measure security throughout the whole system life-cycle. In this part of the thesis, we develop security metrics that target to support system engineers when a system is under design. The metrics that we propose can be categorised as both operational and technical security metrics according to classification given by Sanders [12].

7.6.2 Attack Modelling

Another type of method that quantifies security of a system is based on attack modelling. Although all approaches mentioned above also include some form of attack modelling, this is the main focus of methods from this group. Hence, a lot of details about system behaviour are omitted.

Methodologies for attack modelling have been in a focus of researchers for some time. Weiss [249] describes already in 1991 so-called threat logical trees that resemble modern attack trees. However, the term attack tree was actually coined by Schneier [250, 251] by the end of that decade. A comprehensive survey of attack modelling approaches based on direct acyclic graphs is presented by Kordy et al. [121]. We briefly mention approaches that are not in the scope of this survey, but still relevant to our work.

Almasizadeh and Azgomi [140] define a state-based stochastic model to

quantify different aspects of security, namely Mean-Time to Security Failure (MTSF), attack path probability, and steady-state security. A system is abstracted away as a set of defender transitions, i.e. as backward transitions in an attack model. Semi-Markov chains are employed as a formalism where transitions are associated with uniform distributions. Vigo et al. [252] consider a similar set-up capturing attacker and defender behaviours. However, they adopt a game-theoretical approach for quantification.

Madan et al. [253] propose a method to quantify security attributes of intrusion-tolerant systems (SITAR). The measured quantities are availability of a system and MTSF. All analysis is done based on a single model of the intrusion-tolerant system, i.e. attack and system behaviours, and countermeasure reactions are described within one model. The underlying formalism is a semi-Markov chain. Computations are developed for the steady-state distribution.

The novelty of our approach for quantification of risks is that it explicitly accounts for both elements (system and attacks), but avoids mixing them in a single model specifically created for security analysis. Such an approach has several benefits. A system engineer and a security expert can work separately on the artefacts belonging to their own fields of responsibility and professional expertise. In addition, the approach enables the reuse of self-contained attack models across several systems in one application domain (e.g. different designs of metering devices from the smart grid application domain), because these systems can face the same attacks. It also introduces a new dimension into security analysis – analysis of the impact of a system design itself on security. Thus, our method allows analysing a situation when an attack vector remains the same, but the system itself is modified. Moreover, to our knowledge this is the first method that aims to support system engineers in quantifying risks to data assets in the context of embedded system design models from multiple stakeholder perspectives.

The attack modelling research also focuses on providing methodologies to support elicitation of potential attacks, obtaining sound probabilities to success for attack steps, efficiently propagation these probabilities through the model (from leaves to upper nodes in case of attack trees), and finding more realistic models for representing attack behaviour.

Buldas and Lenin [254] realise that an unrealistic assumption of attack trees is that the steps should be executed in a predefined order, whereas a realistic attack can violate the order, e.g. skipping, reordering, and retrying steps. This work extends the results of Jürgenson and Willemson [255, 256] who deal with effective propagation of probability to success along a tree assuming that all attack steps are attempted independently in parallel.

Arnold et al. [131] provide formalism for time-dependent analysis of attack trees. The measured quantity is the time to success – the probability distribution for a system to be successfully attacked as time progresses. The authors also develop an effective method for evaluation of formalised attack scenarios based on acyclic phase-type distributions.

Philips and Swiler [257] early introduce an idea of an attacker profile as a way to account for attacker capabilities that should be taken into consideration when calculating probabilities of attack steps success. The authors also bring attack templates to facilitate generation of attack graphs for a certain system. These effort has been further extended by other researchers in different contexts, e.g. by LeMay et al. [258], and by Pieters and Davarynejad [259].

LeMay et al. [258] propose a new probabilistic execution model of an attack. Differently from the methods mentioned above where behaviour of attackers is considered to be a Markovian process, LeMay et al. introduce the "look ahead" function that calculates the probability of the current state based on the future state. For this purpose, the authors consider a sophisticated attack model introducing a set of subjective factors to characterise the attack behaviour: attack preferences, goals, and skills. All these factors form an attack profile. The calculated metrics are time to compromise, the probability of taking a certain path, and the probability of being in a certain attack step. Ford et al. [260] present a tool that implements this approach (called ADVISE) in the Möbius Eclipse tool.

Pieters and Davarynejad [259] also investigate how attacker profiles affect probabilities of success for an attack step. The authors distinguish static attacker properties (skills) and dynamic attacker properties (investment schemes). The former is predefined and does not change as an attack is carried out; while the latter is strategically chosen by an attacker and can change over time. These two types of properties when aggregated are given as inputs into a so-called control strength function that gives the probability of success for an attack step.

Atzeni et al. [261] develop an idea of attack personas (adapted from the human-computer interaction research) to support experts in elicitation of potential attacks. An attack persona is a behavioural specification of archetypical attackers on a system.

We believe that such methodologies can be adopted to enrich and strength the attack model component used in our work. In our method we employ a basic attack model concept that captures the minimum elements of an attack model required for our analysis, and thus, giving a certain degree of flexibility for adopting state-of-the-art results in attack modelling.

In line with current attack-based analysis frameworks, our methodology deals with known attacks as opposed to unknown zero-day attacks. Predicting and preparing a system for zero-day attacks is also an important area of research. However, modern systems fail to protect against even known attacks due to lack of security considerations at the design phase. Moreover, Bilge and Dumitras [135] demonstrate that a number of launches of a certain attack when it transients from a zero-day to known attack only grows.

8 Conclusions and Future Work

By coming to the end of the journey, we conclude this thesis and givie an overview on the contributions of this work as well as discuss some interesting directions for future studies and research.

8.1 Conclusions

The increased use of embedded systems in countless applications has led to a boost in their complexity and a need for constant connectivity to open networks. As a result, these systems operate with different categories of data which often include sensitive information. In these realities the task of assuring security properties of embedded systems becomes more challenging which immediately leads to an emergent need in specific methods and tools to support the development of embedded systems.

In this work, we have focused on some aspects of the aforementioned challenge and identified the basic principles currently developed by the research community to address these aspects. These aspects and principles are outlined below. First, the complexity and resource scarcity of embedded systems can no longer sustain the practice of adding security measures at the late development phases. To overcome this issue, the focus on security should be shifted to the design phase following the principles of model-based engineering. Second, although security attracts more attention, there are still a fewer security experts than embedded system engineers. The need to share the knowledge of the security experts and apply it in multiple domains in an effective way emerges. We tackle this issue by implementing the principle of separation of concerns. Finally, complexity of embedded

systems and modern security solutions encourages that security solutions are adjusted to a certain application domain, so that their integration into an embedded system design is less of a burden. We exploit the principle of domain specialisation to address this issue.

To address the named challenge, we have developed an approach called SEED that is built around the basic principles mentioned above. With the help of SEED, security experts gain an opportunity to describe developed security solutions in a reusable manner and embedded system engineers can select a suitable set of security solutions based on an analysis of both system's security needs and its resource constraints.

SEED rests on two concepts introduced in this thesis, namely Domain-Specific Security Model (DSSM) and Performance Evaluation Record (PER). The DSSM and PER concepts described in Chapter 4 are relatively easy to use since they are UML models that are formalised as ontologies: the core security ontology and the core evaluation ontology respectively. These concepts serve as tools for security experts to capture their knowledge about existing security solutions. Each DSSM enables characterising common security issues of a specific application domain in a form of security properties. Thereafter, security properties are linked to available solutions that can be used for their enforcement. Each PER is used to characterise the resource overhead created by a security measure, quality of provided extra-functional properties, and an evaluation technique applied. The underlying ontologies allow storing the knowledge provided by experts and inspecting this knowledge when it is needed for embedded system engineers.

Additionally, the SEED approach is combined with a set of methods and tools (described in Chapter 5 and Chapter 6) that support an embedded system engineer in selecting a suitable set of security measures to be integrated into a system design.

The methods, explained in Chapter 5, assist an engineer to consistently use the knowledge provided by security experts. The first method, called asset elicitation technique, allows analysing a system design and identifying its security needs by consulting DSSMs. The method conducts inspection of functional and execution platform models, represented as SPACE and MARTE models respectively, and consult DSSMs to obtain a set of security properties that are recommended for implementation. A set of security solutions that satisfy security properties are also retrieved from DSSMs. The second method developed in this thesis examines resource constraints of security solutions stored in PERs. This technique, called model-based compatibility analysis, matches platform-related constraints of a system under development and those required by security solutions.

Chapters 4 – 5 also describe a set of tools integrated into the MagicDraw environment as a plug-in that support the SEED approach. These tools use technologies provided by model-driven engineering, e.g. modelling and transformation facilities. Using the metering infrastructure domain as an example, we have shown that the introduced concepts (DSSM and PER)

can be employed by a security expert to describe the security knowledge. Consequently, we have demonstrated that it can be used to support an embedded system engineer to integrate a suitable set of security solutions exploring this security knowledge. This infrastructure has been provided by our industrial collaborators as a use case within the European FP7 SecFutur project.

Next, Chapter 6 focuses on providing a technique for quantifying security risks associated with design models of embedded systems. In particular, we have formalised confidentiality loss and integrity loss as two probabilistic metrics that quantify risks associated with data assets within embedded systems. These metrics reflect both the stakeholder take on assets and exposure of these assets to security breaches. This is achieved by accounting for system design, attack scenarios, and different stakeholder preferences regarding data assets. We have adopted stochastic processes as the computation ground for derivation of these metrics. This allows taking into consideration the time-dependent nature and various uncertainties of involved components. Thereafter, we have extended the losses computed for each asset to a system as a whole. More specifically, we have analysed this case from the standpoint of the expected utility and adapted the form of the function that aggregates confidentiality and integrity loss computed for each asset to a system level.

We have devised two application scenarios for these metrics: they can be used to analyse how certain stakeholders benefit when SBBs are integrated into a system design complementing SEED; and these metrics can be employed to analyse impact of other design decisions on security of a system under development. In addition, we have applied the metrics on a smart metering device and shown their usage for visualising of and reasoning about security risks. Thus, we have illustrated how our methodology allows analysing the impact of design decisions on risks in question, which demonstrates their potential to increase awareness of engineers about security implications for a system at earlier phases.

The SEED approach developed in this thesis contributes to the emerging practice of systematic treatment of security aspects in embedded systems. It is one more step towards providing support to embedded system engineers when designing security-enhanced systems. We conclude with a few reflections about the problem and solution spaces investigated during this work.

Our work rests on the premise that security functions encapsulated into reusable SBBs are basic elements used to support designing of security-enhanced embedded systems. However, it is possible to envisage that enforcement of security properties in an embedded system is not limited to the task of incorporating SBBs. In particular, we find that realisation (at the design phase) of the system functionality itself and a choice of the hardware/software components for an execution platform have also a considerable impact on security properties. For example, if several alternative ways

to allocate security assets on available components exist, the selected allocation may affect security of a system when components differ in terms of their tamper resistance. This, in turn, affects a range of SBBs selected for their integration into an embedded system. While these aspects might be well understood by security experts, there is a shortage of support for embedded system engineers to adopt them when designing a system. We believe that this knowledge possessed by security experts made available for reuse will empower system engineering teams to be more security conscious. Towards this need we have employed the notions of attack scenarios and developed the two metrics of confidentiality and integrity losses that together allow analysing sensitivity of system risk levels with respect to design decisions.

In Chapter 5 we conclude that the use of DSMLs can improve the asset elicitation technique thanks to richer semantics tailored to an application domain. On the other hand, the use of a specific DSML will limit applicability of SEED (or any other approach) confining it to a certain domain. To increase applicability of DSMLs for general approaches is an important challenge. We believe that one way to address it is to elaborate suitable abstraction layers that will allow adapting an approach for a range of DSMLs. To achieve this flexibility we have structured SEED in two levels of abstraction, namely foundation and realisation. The foundation level is the technology-ignorant level that defines basic principles. The realisation level refines the formulated principles according to a selected DSML.

In conclusion, we must mention that there are still other processes where an embedded system is not composed from separate functional and non-functional elements, e.g. building blocks. These processes may have many other emerging challenges when dealing with a task of supporting engineers in developing security-enhanced embedded systems. This is also an interesting direction of work, but it is outside of the scope of this thesis.

8.2 Future Work

There are of course many ways to enrich and build upon the work described in this thesis. In the following, we outline some of them.

8.2.1 Enhancing SEED

The current version of the core security ontology refines the asset concept by two sub-classes, namely data in transit and data stationary. One path for future work is to study what are other important assets to be included into the ontology. These can be refined types of data assets or even asset of different nature, e.g. algorithms, pieces of hardware and software. Security measures can be also considered as assets. However, once new assets are introduced, they should be traceable into a system model to automate their elicitation. The currently employed SPACE modelling language is a general language for modelling of distributed applications. Its semantics

does not allow identifying other classes of assets within functional models specified with this language. To tackle this issue, a more expressive, i.e. more domain-specific, language should be employed for the SEED realisation. This language should allow identifying newly introduced assets automatically from functional and platform models of a system. For example, in the current state of the SEED realisation, pieces of hardware and software can be already identified within a system model since we use MARTE that has appropriate semantics to express embedded system resources. In overall, this direction will require studying both a range of possible assets and existing DSMLs analysing their suitability for extension of the asset elicitation technique.

The representation of security properties is an active and interesting topic. In our work, a tuple consisting of the protected asset, provided security goal, and used defence strategy represents a security property. This convention limits the range of considered security properties to those that can be expressed by such a tuple. However, some security properties can be only expressed, for example, with respect to an operation and an involved actor. Therefore, we think that employment of a more powerful language to express security properties will increase the applicability of the SEED approach. Thus, we propose this topic as another potential direction for future work.

The domain specialisation principle adopted by SEED dictates that security needs of a system vary based on the nature of an application since it imposes different set of assets that can be present in a system. For example, banking application will have transaction records while metering devices will have measurements as one of their typical assets. Besides the need for security protection also depends on present threats imposed by different deployment environments of a system, i.e. the context where a considered embedded system is used. For example, a metering device deployed in a controlled office environment will be exposed to a set of attacks that are different from the attacks possible when the same device is deployed in an opened public environment. The nature of an application is accounted for in SEED by introducing the notion of an application domain. The next step will be to systematically elaborate the threat related element. We envisage two alternative solutions to account for threats when capturing security knowledge. First, one can refine the domain concept including information about context in the definition of the domain itself. For example, instead of specifying a “metering devices” domain, one can define such domains as “metering devices in households” and “metering devices in office buildings”. Another alternative is to encapsulate these considerations in a special threat ontology. In such a way, one can create a set of threat context profiles for one application domain. For example, when there is the metering devices domain, it can be associated with several sets of threat profiles that will characterise threats imposed by different deployment environments, e.g. households, office buildings, and public locations. We find that the latter

alternative gives more flexibility.

In SEED we propose to constantly update the enriched security ontology when capturing the domain-specific security knowledge. Therefore, an important question of maintaining consistency of the enriched ontology arises. In particular, an obvious problem with such an update is pollution of the ontology with concrete SBBs that have different names but refer to the same implementation. Some issues can be resolved by built-in ontology services together with such constructs as *owl:sameAs* or *owl:differentFrom*. However, additional support is needed to ensure that two (or more) concrete SBBs under different names are equal implementations as an example. We envisage that techniques from the area of model comparison or models diff (applied to functional and platform models of concrete SBBs) can be employed to address the mentioned issue.

The starting point of SEED is when an embedded system engineer has an initial version of a functional model of a system and some decisions about a used platform are taken. However, for security-critical systems, e.g. defence applications, the security requirements drive and affect the functionality of a system. Therefore, identifying alternative model-based approaches that start directly from security assets and their security goals can be another subject to study.

The SEED approach provides an infrastructure for capturing diverse performance evaluation results where both resource footprint indices and quality of service characteristics are stored. The natural development of this direction is the use of this data for different kinds of trade-off analyses. Thus, a path for future work will be the exploration of other criteria and strategies to reuse the performance analysis results stored within PERs.

In our work we introduce the notion of model-based compatibility using the hardware resource model defined in MARTE. The natural extension of this course of work is to consider software model for compatibility analysis. The software resource model provides modelling artefacts to describe APIs. This will allow analysing software execution support (API), software services and interfaces.

Finally, availability of advanced and engineer-friendly tools is always a question when it comes to engineering processes. In our work, we use MagicDraw as a platform for our supporting tools since it is selected as an integrating environment within the SecFutur project. However, model-driven engineering increasingly prioritises open standards. Therefore, the migration to such environments as Eclipse Modelling Framework can be a potential thread for future work.

8.2.2 Strengthening Security Metrics

In our work we proposed two metrics for quantification of security risks in design models. These are confidentiality and integrity losses. A natural extension of this direction would be development of models to enable

quantification of the third element of the CIA triad – availability. This will require the definition of an event when data becomes unavailable and mapping it to the formal models.

We envisage that accounting for SBBs and other defences will also naturally fit our work in this area. Obviously, added countermeasures should be accounted as affecting the probability distributions time to success of attack steps. Attack countermeasure trees researched by Roy et al. [262, 263], defence trees proposed by Bistarelli et al. [264], and attack-defence trees studied by Kordy et al. [265, 266] are examples of such formalisms. However, these integrated SBBs can also influence execution time distributions at system states since added security features may require extra computations. The main challenge in this part will be to provide such a means for adjusting the execution time and time to success probability distributions when a certain SBB is selected by a system engineer. Naively, one can think about re-doing the whole analysis for obtaining execution time and time to success. However, a natural approach would be treating this as compositional security analysis, when integration of a SBB corresponds to updating system and attack models with a new component. In this vein, the research question can be formulated as follows: given the results of the earlier assessment, how to reuse them by adjusting these results when a system is updated with a new component (a SBB) without repeating time- and resource-consuming assessments?

We have formalised a system model as a semi-Markov chain and mainly rely on transient probabilities when defining confidentiality and integrity losses. Exploitation of other Markovian statistics to provide even more elaborate and expressive definitions of risks to data assets can also be a viable direction for future works. For example, using the already formalised models one can obtain the number of times that the process enters a certain state through a given time interval. This, in turn, can be used to enhance the cost functions that may be sensitive to the number of times a confidential data item has been observed by an attacker as an example. Another interesting measure is a first passage time that is a measure of how long it takes to reach a given state from another state. We believe that such statistics can complement the current state of our method.

In our work, we extensively use the cost and full cost functions for expressing consequences when a data asset is attacked. However, it is not a trivial task to construct these functions. To assign these costs stakeholders may need to consider business values of the company and its strategy. We can envisage that these cost functions can be built as a result of interviews and workshops with involved parties, and by automatically profiling asset users [124]. We think that construction of an effective and sustainable method to obtain adequate cost functions is an interesting direction for future works.

Attack process modelling is central to security research. In common with many current attack-based analysis frameworks, our methodology deals with

known attacks as opposed to unknown zero-day attacks. While it is equally important to prepare a system to function under known attacks, detection of unknown attacks is also a research challenge. The methodology proposed in our work will only benefit when zero-day attacks are also considered.

Finally, we contribute to support a system engineer by defining two metrics for quantifying security risks to data assets. This is an important step, but still only a small part of the big problem of securing our systems and infrastructures. To achieve better security it should be measured from various angles. Therefore, we believe that researchers and practitioners should aim for constructing a holistic security measuring infrastructure (eco-system). This framework should enable measuring security throughout the whole lifecycle and on all organisational levels.

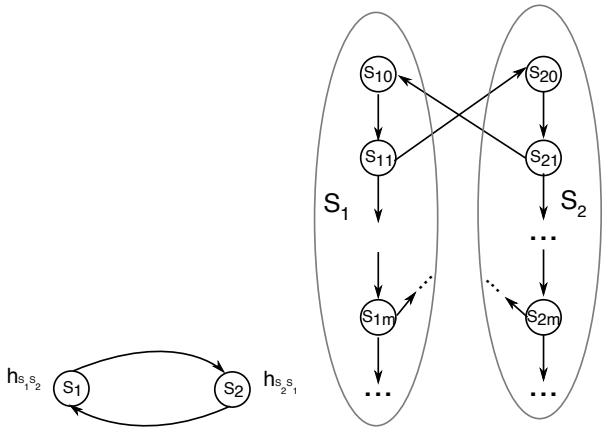
A

Semi-markov Chain Approximation

The main difference between Semi-Markov Chains (SMCs) and Discrete-time Markov Chains (DTMCs) is that each state of a SMC is additionally annotated with holding time distributions [76]. In this work, we have used a transformation that approximates a SMC as a DTMC. The obtained DTMC can be then computed using such powerful model checkers as PRISM [133]. This, in turn, enables conducting transient analysis for SMCs required for computation of the confidentiality and integrity loss metrics defined in Chapter 6.

Proposed approximation

Figures A.1(a) and A.1(b) illustrate an idea of the proposed approximation. A SMC depicted in Figure A.1(a) has two states s_1 and s_2. The SMC moves to state s_2 from s_1 with the transition probability $p_{s_1 s_2}$ and to state s_1 from s_2 with the transition probability $p_{s_2 s_1}$. However, before making a transition the process holds for some time $\tau_{s_1 s_2}$ in a current state. This holding time for each state is given by assigning a probability mass function that expresses the time that a system will stay in a state before proceeding to a next state as exemplified in Figure A.1(a). In our example, $h_{s_1 s_2}(t)$ and $h_{s_2 s_1}(t)$ are holding times in states s_1 and s_2 respectively. Note that in general case holding time can depend on the next transition, therefore, the notation for holding times contains both the initial state and the destination. Hence, $h_{s_1 s_2}(t)$ is interpreted as holding time for a process in state s_1 given that



(a) Original SMC (b) Transformed DTMC

Figure A.1: Overview of the proposed approximation

the next transition is taken into state s_2. Thus,

$$P(\tau_{s_is_j} = t) = h_{s_is_j}(t),\ i = 1, 2, \ldots, N; j = 1, 2, \ldots, N; t = 1, 2, \ldots. \quad \text{(A.1)}$$

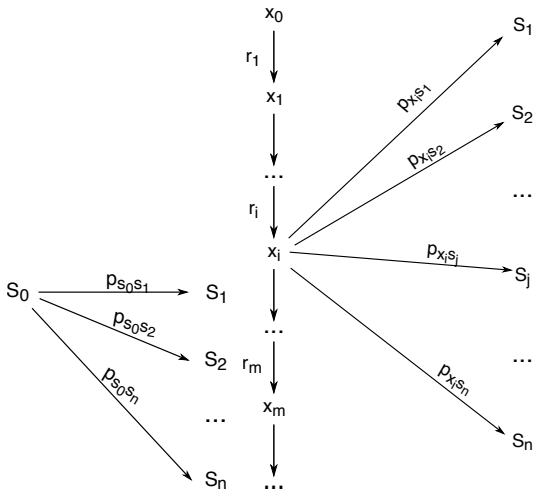
We consider a case where holding time distributions are discrete distributions. There is also an assumption that all holding times are at least one time unit in length, i.e. $h_{s_is_j}(0) = 0$.

We propose to construct an equivalent DTMC by expanding each origin state in a SMC by a set of DTMC states that emulate the holding process as exemplified in Figure A.1(b). This DTMC has two sets of states. The states $s_{10}, s_{11}, ..., s_{1m}, ...$ correspond to an original state s_1 and the states $s_{20}, s_{21}, ..., s_{2n}, ...$ correspond to an original state s_2. The states in each of such sets are internally related by "internal" transitions. In particular, there is a transition for each pair of states s_{1i} and $s_{1(i+1)}$ (s_{2i} and $s_{2(i+1)}$). Additionally, since there is a transition from s_1 to s_2 in the original SMC, each state s_{1i} has a transition to s_{20} in the approximating DTMC. Similarly, each state s_{2i} has a transition to s_{10}.

The proposed approximation can be understood with the following intuition. For example, let us assume that the original SMC decides to stay for τ time units in state s_1. The corresponding behaviour of the DTMC from Figure A.1(b) is that it moves from state s_{10} to state $s_{1\tau}$ along $s_{1i} : 0 < i < \tau$ states only. Now let us assume that the original SMC decides to move from s_1 to s_2 after holding in s_1 for τ time units. This behaviour is emulated by the approximating DTMC as a move into $s_{1(\tau-1)}$ from s_{10} along $s_{1i} : 0 < i < \tau - 1$ and then taking the transition from $s_{1(\tau-1)}$ to s_{20}.

Formalisation

We consider a SMC as shown in Figure A.2(a). This model has $n + 1$ states. Table A.1 shows transition probabilities for this model (see row 2) and holding time distributions (see row 3). We say that each $h_{s_0 s_i}(t)$ is a vector $\{a_1^i, a_2^i, \ldots, a_m^i, \ldots\}$ where $a_\tau^i = h_{s_0 s_i}(\tau)$ for any valid $\tau = 1, 2, \ldots$. For simplicity, we also assume that $h_{s_0 s_1}(\tau) = h_{s_0 s_2}(\tau) = \cdots = h_{s_0 s_n}(\tau) = a_\tau$ for any valid $\tau = 1, 2, \ldots$.



(a) Original SMC (b) Approximating DTMC

Figure A.2: Illustration of the proposed approximation

Table A.1: Transition probabilities and holding time distributions

	s_1	s_2	...	s_n
transition probabilities from s_0	$p_{s_0 s_1}$	$p_{s_0 s_2}$	...	$p_{s_0 s_n}$
holding time distributions for s_0	$h_{s_0 s_1}(t)$	$h_{s_0 s_2}(t)$	...	$h_{s_0 s_n}(t)$

A DTMC approximated from the SMC is shown in Figure A.2(b). This figure shows the approximation applied only for the state s_0. In the approximating DTMC model, the original state s_0 from the SMC is replaced by an additional set of states $x_0, x_1, \ldots, x_m, \ldots$. Moreover, new transitions from each x_i to s_j (denoted as $p_{x_i s_j}$), and from each x_i to $x_{(i+1)}$ (denoted as r_{i+1}) are added. The transition probability matrix for this DTMC model is summarised in Table A.2. In this table, r_i and $p_{x_i s_j}$ are calculated according

to equations (A.2) and (A.3) respectively.

$$r_i = \begin{cases} 1 - a_1, i = 1 \\ 1 - \dfrac{a_i}{\prod_{k=1}^{i-1} r_k}, otherwise \end{cases} \tag{A.2}$$

$$p_{x_i s_j} = (1 - r_{i+1})\, p_{s_0 s_j} \tag{A.3}$$

Recall that a_τ is a probability of holding in a current state for τ time units. Derivation of these expressions are explained in the last rubric of this appendix.

Table A.2: Transition probabilities for a resulting DTMC

	x_0	x_1	x_2	...	x_m	...	s_1	s_2	...	s_n
x_0	0	r_1	0	...	0	...	$p_{x_0 s_1}$	$p_{x_0 s_1}$	...	$p_{x_0 s_n}$
x_1	0	0	r_2	...	0	...	$p_{x_1 s_1}$	$p_{x_1 s_1}$	...	$p_{x_1 s_n}$
...	...	...	...	...	...	...	...	...	...	...
x_m	0	0	0	...	0	...	$p_{x_m s_1}$	$p_{x_m s_1}$	...	$p_{x_m s_n}$
...	...	...	...	...	...	...	...	...	...	...

Preliminary results

We apply the proposed approximation to a SMC that has 15 states. The transition matrix is depicted in Table A.3. Table A.4 shows holding time distributions (probability mass functions) in each state. We assume that holding time distributions do not depend on the next transition. In some literature such distributions are referred to as waiting time.

Table A.3: Transition probabilities for an example system

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}
s_0	0	0.03	0.01	0.96	0	0	0	0	0	0	0	0	0	0	0
s_1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s_2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s_3	0	0	0	0	0.02	0.98	0	0	0	0	0	0	0	0	0
s_4	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
s_5	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
s_6	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
s_7	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
s_8	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
s_9	0	0	0	0	0.13	0	0	0	0	0	0.87	0	0	0	0
s_{10}	0	0	0	0	0	0	0	0	0	0	0	0.03	0	0.97	0
s_{11}	0	0	0	0	0.25	0	0	0	0	0	0	0	0.75	0	0
s_{12}	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
s_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
s_{14}	0	0	0.01	0.01	0.98	0	0	0	0	0	0	0	0	0	0

Table A.4: Holding time distributions for an example system

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
holding time distribution	$\mathcal{N}(10,3)$	$\mathcal{N}(50,8)$	$\mathcal{N}(50,8)$	$\mathcal{N}(63,8)$	$\mathcal{N}(155,15)$	$\mathcal{N}(12,3)$	$\mathcal{N}(8,3)$	$\mathcal{N}(14,3)$
	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	
holding time distribution	$\mathcal{N}(10,3)$	$\mathcal{N}(100,2)$	$\mathcal{N}(45,7)$	$\mathcal{N}(3,1)$	$\mathcal{N}(10,3)$	$\mathcal{N}(12,3)$	$\mathcal{N}(10,3)$	

When the proposed approximation has been applied, we obtain a DTMC model that has 1100 states. Thereafter, we compute transient probabilities for the DTMC model in PRISM [133] and compare these results obtained by solving the original SMC model (equation 2.2) analytically. We take a time interval t of 1000 units.

The results are depicted in Figure A.3. This figure shows that there is only a slight divergence (0.46%) of the analytically obtained results from the ones obtained from the approximating DTMC.

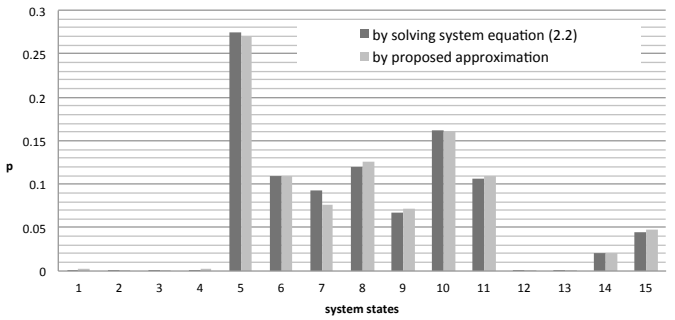


Figure A.3: Preliminary results

In general case the used probability mass functions for holding time of states can have a long tail (countably infinite). However, for the implementation of the proposed approximation, we assume that the vector $h_{s_0 s_i}(t)$ is actually finite. This means that starting from some time point $\tau : \tau \leq t$ we assume that $h_{s_0 s_i}(t) = 0$. We refer to this truncated version of $h_{s_0 s_i}(t)$ as $h'_{s_0 s_i}(t)$. To deal with the remaining tail, we add the aggregated probability into the last element of the new vector $h'_{s_0 s_i}(t)$. In particular,

$$h'_{s_0 s_i}(\tau) = 1 - \sum_{k<\tau} h_{s_0 s_i}(k) \tag{A.4}$$

Obviously, this is not exact and the error should be estimated.

A rationale of cutting off this tail is an insight that an exact approximation can lead to a large number of newly added states x_i. This, in turn, leads to the state explosion problem. We envisage that when there are tools

to estimate the introduced error, one can balance the needed precision and the cost of computations.

Derivation of transition probabilities

In this section we explain derivation of the expressions for r_i (shown in system of equations (A.2)) and $p_{x_i s_j}$ (shown in equation (A.3)). We focus on the SMC and its approximating DTMC that are depicted in Figure A.2(a) and Figure A.2(b) respectively.

The validity of equation (A.2) for r_i lies in solving the following system of equations:

$$\begin{cases} r_1 = 1 - a_1 \\ r_1 r_2 = 1 - a_2 - a_1 \\ r_1 r_2 r_3 = 1 - a_3 - a_2 - a_1 \\ \dots \\ r_1 r_2 r_3 \dots r_m = 1 - a_m - \dots - a_3 - a_2 - a_1 \end{cases} \tag{A.5}$$

The rationale behind this system of equations is explained as follows: if a system proceeds along the transition r_1 (in the approximating DTMC from Figure A.2(b)), it means that it will stay in s_0 for $t > 1$ (in the SMC from Figure A.2(a)). In other words,

$$r_1 = P(\xi > 1) = 1 - P(\xi \leq 1) = 1 - (h_{s_0 s_i}(1) + h_{s_0 s_i}(0)) = 1 - a_1 \tag{A.6}$$

Similarly, if a system proceeds along the transition from r_1 to r_2, it means that it will stay in s_0 for $t > 2$. Thus,

$$r_1 r_2 = P(\xi > 2) = 1 - P(\xi \leq 2) = 1 - \sum_{k=0}^{2} h_{s_0 s_i}(k) = 1 - a_2 - a_1 \tag{A.7}$$

The same logic is applied up to the r_m state to obtain the system of equations (A.5). We exclude a_0 since $a_0 = h_{s_0 s_j}(0) = 0$.

Now, we show that equation (A.3) for $p_{x_i s_j}$ exhibits the behaviour of an approximating DTMC that is equal to the behaviour of an original SMC. Recall that we focus on the SMC and its approximating DTMC that are depicted in Figure A.2(a) and Figure A.2(b) respectively.

We know that one of the ways to come to s_i from s_0 by t is to hold in s_0 for t and then take the transition to s_i. This behaviour can be expressed as:

$$\phi^{SMC}_{s_0 s_i}(t) = h_{s_0 s_i}(t)\, p_{s_0 s_i} \tag{A.8}$$

In terms of the approximating DTMC, this transition from s_0 to s_i corresponds to moving into state x_{t-1} from x_0 and then taking the transition from x_{t-1} to s_i. This behaviour can be expressed as:

$$\phi_{s_0 s_i}^{DTMC}(t) = \prod_{k=1}^{t-1} r_k \; p_{x_{t-1} s_i} \tag{A.9}$$

where $p_{x_{t-1} s_i} = (1 - r_t) \; p_{s_0 s_i}$ according to equation (A.3). To check whether the SMC and the DTMC exhibit similar behaviour, we need to show that $\phi_{s_0 s_i}^{SMC}(t) = \phi_{s_0 s_i}^{DTMC}(t)$. That is, we need to show that

$$h_{s_0 s_i}(t) = \prod_{k=1}^{t-1} r_k \; (1 - r_t) \tag{A.10}$$

This is straightforward by replacing r_t with the expression from equation (A.2).

Note that the expression $p_{x_i s_j} = (1 - r_{i+1}) \; p_{s_0 s_j}$ also yields correct properties with respect to transitions outgoing from one state. In particular,

$$r_m + (1 - r_m) \sum_{j=1}^{n} p_{x_{(m-1)} s_j} = 1 \tag{A.11}$$

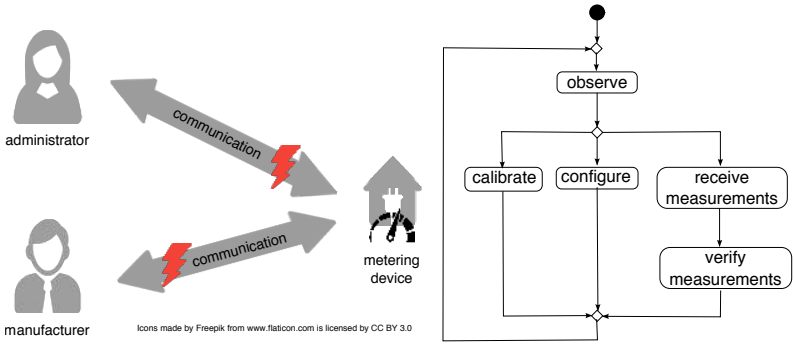
B
Scenario Setup

Figure B.1(a) describes the setup for our scenario used in Chapter 6. There are four main components: a manufacturer, a metering device, an administrator, and a collector. Operations of these components imitate a setup designed for the TSN infrastructure [3] within the SecFutur project [17].

A manufacturer represents an organisation that produces the device and is also responsible for its maintainance. This actor can connect to the metering device to provide or revoke guarantees that this device is produced according to (legal) specifications. These guarantees are stored on the device in a form of a manufacturer certificate. In our scenario, the manufacturer talks to a metering device once per year. An administrator combines functions of a calibrator and a configurator. Thus, the administrator is responsible for configuring and for calibrating the meter and its sensors according to certain rules. For this scenario, we set a configuration period to three times per year and a calibration period to twelve times per year. A collector collects measurements received from metering devices, verifies, and acknowledge these measurements. Both the collector and the administrator are implemented as one component. Figure B.1(b) shows basic actions for this component that reflect functions of the collector and the administrator. The metering device is described in Section 6.3.

For our prototype, we use the hardware and firmware provided by the open-source project OpenEnergyMonitor [1]. The manufacturer, the administrator and the collector are implemented on a Raspberry Pi platform [2]

[1] `http://www.openenergymonitor.org/emon/`
[2] `http://www.raspberrypi.org`



(a) Main actors

(b) The administrator component (simplified UML activity)

Figure B.1: Scenario setup used for the prototype

equipped with an additional radio board (RFM12B [3]). The metering device is implemented based on the Arduino platform [4] assembled with the same radio module. The measurement eavesdropping and spoofing attacks are also implemented on the same Raspberry Pi platform.

To obtain the probability distributions used in Chapter 6 we conducted 100 experiments for each metering device state and measured its execution time. Thereafter, we calculated mean and standard deviation and used these two values as the corresponding parameters for a normal distribution. A similar approach was used for obtaining time to success probability distributions for the two attacks mentioned above. However, we applied the Kernel Density Estimation (KDE) by finding the curve that fitted best the shape of our samples (by adjusting the bandwidth parameter). For this purpose, we used the Python SciPy [5] library implementation of the KDE (the gaussian_kde function).

[3] http://www.hoperf.com/rf/fsk_module/RFM12B.htm
[4] http://www.arduino.cc
[5] http://www.scipy.org

C

From Engineering Artefacts to Formal Models

In this part, we explain how the formal models introduced in Section 6.2, i.e. data and state models, have been obtained from engineering artefacts. We focus on UML models used for the SEED realisation. In particular, we consider UML activity models as a functional model (and its slight modification SPACE) and UML class models annotated with stereotype from MARTE::HRM as an execution platform model.

Concrete syntax

In this section, we outline some parts of UML activities and MARTE concrete syntax. This provides basics to assist the reader in creating a link between syntactical constructs and abstract syntax, and its mapping onto the formal data and state models.

In general, an (UML) activity model can be built of activities and actions where each activity is built of actions. For the sake of simplicity and without loss of generality, we assume that there is only one activity since nested activities is just a syntactic sugar and they can be flattened. Actions and activities are related to each other by directed edges that create two types of flows: a control flow and an object flow. Intuitively, a control flow determines a logical flow of activities/actions and an object flow carries data objects. Similar to SPACE, we require that only one edge can leave and arrive to an action. Control and data flow edges can be connected to special types of nodes that are called control nodes. These are decision, merge, join, and fork nodes. They serve for coordinating flows in an activity.

UML 2.4.1 specifies two syntactical constructions to express an object flow, namely as pins attached to outgoing and incoming nodes, and as an object node associated with an activity edge. SPACE adopts the pin notation. Note that in the standard [28] an object node associated with an activity edge implies that this edge is split into two object flows: incoming and outgoing. For simplicity we converge both notations to one activity edge, that is an object flow, associated with an object. Usually, an engineer does not need to create explicitly a special edge to denote an object flow, but rather he/she annotates an existing control flow with an object.

UML does not define explicitly means for assigning transition probabilities to edges and execution time distributions to actions. However, this information can be easily added into activity models by using the comment structure. Note that since we assume that only one edge can leave an action, a system engineer only needs to explicitly annotate with probabilities those edges that are outgoing from decision control nodes.

As a model of an execution platform we consider a UML class model annotated with stereotypes from MARTE::HRM (e.g. as depicted in Figure 5.4). In general, a wide range of structural UML elements can be annotated with these stereotypes. For the sake of this explanation, we will consider a small part of UML syntax. Hence, we assume that a component of an execution platform is represented as a class. The structure of an execution platform is represented by relating classes to each other with association links. These links can capture different kinds of relations between components, but we omit these details in this section.

The allocation information can be captured in models using the concepts from the MARTE::Alloc package. In particular, we use the allocation model defined in this package. This model serves to map the application model elements (logical parts) to the execution platform elements (physical parts). There are several syntactical constructs to capture an allocation link that provide various notations. A system engineer can use the *allocate* or *assign* stereotypes. We consider this variety as syntactic sugaring. An allocation link has two ends: source and target. In case of the SEED realisation, the source is a behaviour element of a functional model (action) and target is a structural element of an execution platform.

Abstract syntax

Now we formalise the concrete syntax outlined above to enable definition of a semantic function that maps UML models of a system into the formal models introduced in Section 6.2.2.

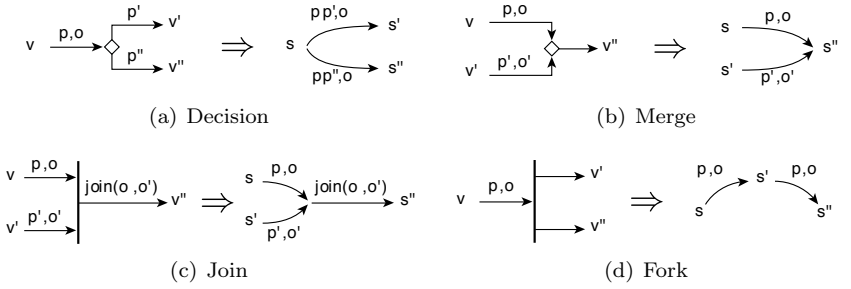
We say that a UML system model (Y_{UML}) is built of two elements, namely a functional model (FM) and its execution platform model (EP). A functional model FM is a tuple (v_0, V, E, P, l_{ex}) where the first three elements compromise a graph, $P : E \to [0, 1]$ is a function that associates probabilities to edges, and $l_{ex} : V \to \mathcal{F}$ is a labelling function that associates execution time probability distributions to each activity node V. An

execution platform model EP is a tuple (C, l_{al}) where C is a set of components and $l_{al} : V \rightarrow 2^C$ is an allocation function that associates activity nodes with components of an execution platform. We also use the following helper functions:

- Two functions mapping an activity node and edge to their particular types, namely $kind_V : V \rightarrow K_V$ and $kind_E : E \rightarrow K_E$, where $K_V = \{executable, merge, join, fork, decision, timer, other\}$ and $K_E = \{object, control\}$.
- Two functions that return the source and target nodes of a given edge, i.e. $source : E \rightarrow V$ and $target : E \rightarrow V$ respectively.
- A function that returns an object flowing along a certain edge, namely $getobj : E \rightarrow O$ where O is a set of object nodes.

Semantic Mapping

Transformation of a UML activity diagram into a state machine is the ongoing research subject [125, 267]. We capitalise on this research area. Briefly, this transformation can be summarised as follows. Each action node from a UML activity diagram is transformed into a state; and transitions from an activity diagram are directly transformed into transitions between states. Control nodes, i.e. decision, merge, join, and fork, are transformed employing a set of rules. Figure C.1 summarises the basic transformation rules for control and object flows when a control node is encountered. We develop these rules from studies conducted by Ouchani et al. [125], Herrmann and Kraemer [267], and Störrle [268], and from the semantic interpretation of control nodes provided by UML 2.4.1 [28]. In this appendix, we assume that there are no nested control nodes. If such cases exist we assume that there is a function *removeNested* that flattens such constructions. Detail elaboration of this function is outside of the scope of this appendix.



(a) Decision (b) Merge

(c) Join (d) Fork

Figure C.1: Transformation rules for control nodes

In this section, we describe a semantic mapping of *FM* and *EP* (an UML system model) into *SD* and *DM* (the formal system model introduced in Section 6.2) by a function $[\![\cdot]\!]$ adopted for our case studies. The semantic function $[\![\cdot]\!]$ is defined as follows:

- *Initial state.* An initial state s_0 of *SM* is a direct mapping of an initial state v_0 of *FM*, i.e. $s_0 = v_0$.
- *System states.* A set of states S of *SM* is built of two subsets S_1 and S_2 as $S = S_1 \cup S_2$. These two subsets are formed as follows:
  - The subset S_1 includes all executable and timer vertices from V, namely $S_1 = \{v \mid v \in V, kind_V(v) \in \{executable, timer\}\}$.
  - The subset S_2 includes vertices that correspond to the join control node (observe state s_j in Figure C.1(c)), namely $S_2 = \{v \mid v \in V, kind_V(v) = join\}$
- *Transitions and transition probabilities.* To obtain the transition matrix P, we need to transform control flows of *FM*. This flow is governed in *FM* by a set of control nodes. To deal with these constructs of UML activity models we need to consider five basic cases. These cases are analysed below.

  1. In the trivial case there are two vertices in *FM* connected directly by an edge $e \in E$. This edge e is directly transformed into a relation between corresponding states in *SM* by assigning the same probability $P(e)$ to $P(s, s')$. Note, that $P(s, s') = 0$ if such an edge e does not exist. Thus,

$$\begin{aligned} &P(s, s') = P(e) \text{ where} \\ &e \in E,\ [\![v]\!] = s,\ v = source(e), \\ &[\![v']\!] = s',\ v' = target(e) \end{aligned}$$

  2. A decision node is a node that selects between its outgoing flows. If there is a decision node v_d between two vertices v and v' in *FM* then the mapping function needs to collapse two edges e (connecting v and v_d) and e' (connecting v_d and v') into one relation between corresponding states s and s' in *SM*. This case is illustrated in Figure C.1(a) and can be expressed as follows:

$$\begin{aligned} &P(s, s') = P(e) \cdot P(e') \text{ where} \\ &e \in E, e' \in E,\ v_d = target(e), \\ &kind_V(v_d) = decision,\ [\![v]\!] = s,\ v = source(e), \\ &v_d = source(e'),\ [\![v']\!] = s',\ v' = target(e') \end{aligned}$$

3. A merge control node brings together multiple alternate flows. If there is a merge node v_m between two vertices v and v'' in FM then the mapping function needs to collapse two edges e (connecting v and v_m) and e_j (connecting v_m and v'') into one relation between the corresponding states s and s''. This case is illustrated in Figure C.1(b) and can be expressed as follows:

$$
\begin{aligned}
&P(s, s'') = P(e) \text{ where} \\
&e \in E, e'' \in E,\ v_m = target(e), \\
&kind_V(v_m) = merge,\ [\![v]\!] = s,\ v_i = source(e), \\
&v_m = source(e''),\ [\![v'']\!] = s'',\ v'' = target(e'')
\end{aligned}
$$

4. A join control node synchronises the incoming flows. If there is a join node v_j between two vertices v and v'' in FM then the mapping function creates two relations between the corresponding mapped states s, s_j, and s''. This case is illustrated in Figure C.1(c) and can be expressed as follows:

$$
\begin{aligned}
&P(s, s_j) = P(e) \text{ where} \\
&e \in E,\ [\![v_j]\!] = s_j,\ v_j = target(e), \\
&kind_V(v_j) = join,\ [\![v]\!] = s,\ v = source(e)
\end{aligned}
$$

and

$$
\begin{aligned}
&P(s_j, s'') = P(e') \text{ where} \\
&e \in E, e'' \in E, [\![v_j]\!] = s_j, v_j = target(e), \\
&kind_V(v_j) = join, [\![v]\!] = s, v = source(e), \\
&kind_V(v) \in \{executable, timer\},\ [\![v'']\!] = s'', \\
&v'' = target(e''),\ v_j = source(e'')
\end{aligned}
$$

5. A fork node splits a flow into multiple concurrent flows and actions of these flows are carried out in parallel. The semantic function maps these parallel flows to an interleaved sequence of states (which is shown to be a correct refinement by Kraemer and Herrmann [267]). For simplicity, we demonstrate the mapping function that splits the flow into two parallel flows. This case is illustrated in Figure C.1(d) and can be expressed as follows:

$$P(s, s') = P(s', s'') = P(e) \text{ where}$$
$$e' \in E, e'' \in E,\ kind_V(v_f) = fork,\ [\![v]\!] = s,$$
$$v = source(e),\ v_f = target(e),\ v_f = source(e'),$$
$$v_f = source(e''),\ [\![v']\!] = s',\ v' = target(e'),$$
$$[\![v'']\!] = s'',\ v'' = target(e'')$$

Note that when a fork vertex is encountered it affects the rest of the transformation since all subgraphs starting from this fork vertex should be interleaved. We refer to this interleaving procedure as *forkEffect*.

- *Holding time.* A holding time distribution for a state $s \in S$ from *SM* is a direct mapping of execution time for a corresponding vertex in *FM*. Thus, $\{H(s) \mid s \in S\} = \{l_{ex}(v) \mid v \in V, [\![v]\!] = s\}$. A holding time distribution for state s_j (a node created when a join node is encountered) is a distribution of inter-arrival time between the first and the last objects (o and o') arrived into a corresponding join node.

- *Data dependencies.* We say that two distinct data objects o and o' are immediately dependent, and therefore, $(o, o') \in D$ if there is such a vertex v in *FM* that o is its incoming object and o' is its outgoing object.

$$D = \{(o, o') \mid v \in V,\ [\![v]\!] = s,\ e, e' \in E,$$
$$v = target(e),\ kind_E(e) = object,$$
$$o = getobj(e),\ v = source(e'),$$
$$kind_E(e') = object,\ o' = getobj(e')\}$$

- *Labelling functions.* Finally, we need to provide the mapping for three labelling functions defined for *SM* and *DM*, i.e. l_C, l_O, and l_D.

  - The labelling function l_C is easily obtained from the allocation l_{al} from *EP* by direct mapping. In particular, a state $s \in S$ is associated with a set of components $C' \subset C$, i.e. $l_C(s) = C'$, where for the corresponding vertex $v \in V : l_{al}(v) = C'$. Thus,

    $$l_C(s) = \{c \mid c \in C, v \in V, [\![v]\!] = s, c \in l_{al}(v)\}$$

  - To obtain the labelling function l_O, we say that all objects coming in and outgoing from a vertex v along activity edges in *FM* are associated with a corresponding state s in *SM* as objects associated with this state. Thus,

$$
\begin{aligned}
l_O(s) = \big\{ & \{o, o'\} \mid v \in V,\ [\![v]\!] = s,\ e \in E, \\
& e' \in E,\ v = target(e),\ kind_E(e) = object, \\
& o = getobj(e),\ v = source(e'), \\
& kind_E(e') = object,\ o' = getobj(e') \big\}
\end{aligned}
$$

– Finally, for any dependency $(o, o') \in D$ the labelling function l_D is defined as follows:

$$
\begin{aligned}
l_D(o, o') = \{ & s \mid [\![v]\!] = s,\ v \in V,\ e \in E, \\
& e' \in E,\ v = target(e),\ v = source(e'), \\
& kind_E(e) = object,\ kind_E(e') = object, \\
& o = getobj(e),\ o' = getobj(e') \}
\end{aligned}
$$

The semantic mapping function introduced above assumes the existence of dependencies between two distinct objects that enter and leave the same node. However, UML does not explicitly define such semantics. Therefore, the DM model obtained after applying $[\![\cdot]\!]$ should be additionally reviewed by an engineer to exclude incorrectly assumed dependencies. Figure C.2 depicts the process for transforming of a UML system model, i.e. $Y_{UML} = (FM, PM)$, into a formal model, i. e. $Y = (SM, DM)$, introduced in Section 6.2.2.

Figure C.2: Application of the semantic function to a UML system model

D
Experiment Details

In this appendix we give additional details of the computations discussed in Chapter 6. Figure D.1 illustrates inputs used for our example and its respective outputs.

There are three types of input elements. These are system design models, attack models, and stakeholder estimates (see equation 6.4). In Chapter 6 we used two design alternatives (original and modified designs), two attacks (eavesdropping and spoofing), and 30 stakeholder cost estimates (three stakeholders, two security goals, and five data objects). Two mentioned system designs and attacks are described in Sections 6.3.1 and Section 6.3.2 respectively. Additionally, Tables A.3 and A.4 summarise transition and holding time probabilities (shown in Figure 6.6) in a matrix form. Table D.1 complements Table 6.3 by giving details about stakeholder cost estimates for all data objects involved into the example from Section 6.3. To calculate confidentiality loss, we combine each design model with the eavesdropping attack. Analogously we combine a system model with the spoofing attack when computing integrity loss.

Figure D.2 contains four subfigures that depict calculated values for risks to confidentiality and integrity of measurements over time. There are fluctuations in the *CL* and *IL* values in the beginning (up to 17s for *CL* and up to 10s for *IL* in Figures D.2(a) and D.2(b) respectively) and, thereafter, the curves are stabilising or experiencing only small oscillations. Technically this behaviour corresponds to stabilisation of underlying SMCs to their steady-states that is the long run behaviour. We use values observed after the stabilisation points for the figures discussed in Section 6.3.3, i.e. Figures 6.8 and Figure 6.9. In particular, these points are 17s for *CL* and 10s for *IL* in

case of the original design (e.g. Figure 6.8(a)) and 443 for *CL* and for *IL* in case of the modified design (e.g. Figure 6.8(b)).

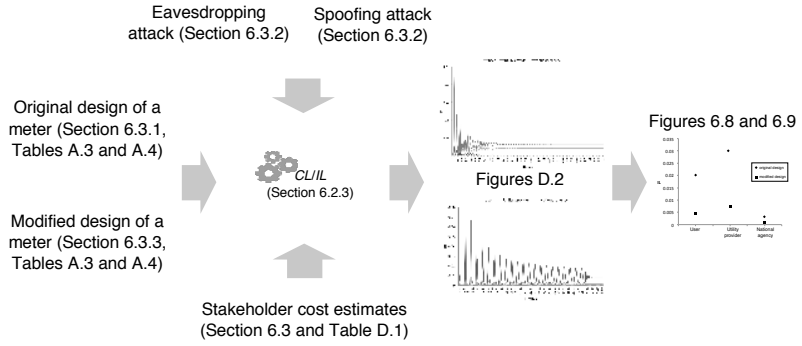
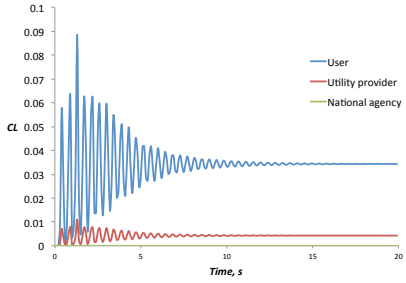
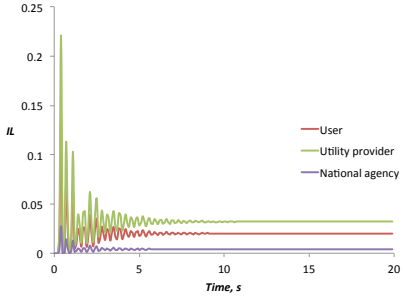


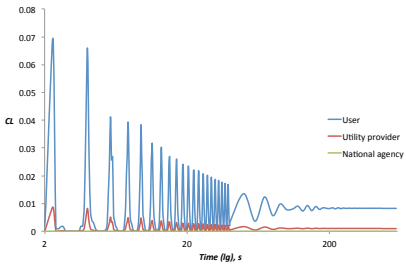
Figure D.1: Illustration of input and output data



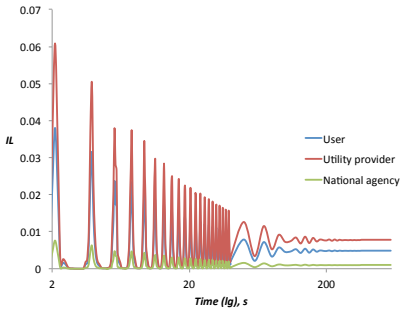
(a) Original design, confidentiality loss



(b) Original design, integrity loss



(c) Modified design, confidentiality loss



(d) Modified design, integrity loss

Figure D.2: Visualisation of calculated *CL* and *IL*

Table D.1: Stakeholder cost estimates (I – Integrity and C – Confidentiality)

Asset	Goal	User	Utility provider	National agency
Measurements	I	moderate (0.5)	major (0.8)	minor (0.1)
	C	major (0.8)	minor (0.1)	insignificant (0)
Commands	I	insignificant (0)	moderate (0.5)	major (0.8)
	C	insignificant (0)	moderate (0.5)	minor (0.1)
Raw measurements	I	minor (0.1)	minor (0.1)	insignificant (0)
	C	moderate (0.5)	minor (0.1)	insignificant (0)
Measurement ids	I	insignificant (0)	minor (0.1)	insignificant (0)
	C	insignificant (0)	minor (0.1)	insignificant (0)
Acknowledgements	I	insignificant (0)	insignificant (0)	insignificant (0)
	C	insignificant (0)	insignificant (0)	insignificant (0)

Bibliography

[1] Markus Voelter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart C. L. Kats, Eelco Visser, and Guido Wachsmuth. *DSL Engineering – Designing, Implementing and Using Domain-Specific Languages*. Dslbook.org, 2013.

[2] Frank Alexander Kraemer. *Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks*. PhD thesis, Norwegian University of Science and Technology, August 2008.

[3] Deliverable D2.1b: Documentation of Use Cases, Requirements and Success Factor Indicators. EU SecFutur project, 2012.

[4] Charles Q. Choi. Cisco IP Phones Vulnerable. `http://www.spectrum.ieee.org`, 2013, 2013.

[5] Peter Clarke. Embedded Systems Next for Hack Attacks. `http://www.eetimes.com`, 2013.

[6] Srivaths Ravi, Anand Raghunathan, Paul Kocher, and Sunil Hattangady. Security in Embedded Systems: Design Challenges. *ACM Transactions on Embedded Computing Systems*, 3(3):461–491, August 2004.

[7] Stephanie Neil. Securing Devices By Design. `http://www.automationworld.com`, 2015.

[8] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 2012.

[9] Paul Kocher, Ruby Lee, Gary McGraw, and Anand Raghunathan. Security as a New Dimension in Embedded System Design. In *Annual Design Automation Conference (DAC)*, pages 753–760. ACM, 2004.

[10] Peter Skaves. FAA Aircraft Systems Information Security Protection Overview. In *Integrated Communication, Navigation, and Surveillance Conference (ICNS)*, pages A1–1–A1–17. IEEE, 2015.

[11] David Basin and Srdjan Capkun. The Research Value of Publishing Attacks. *Communications of the ACM*, 55(11):22–24, 2012.

[12] William H. Sanders. Quantitative Security Metrics: Unattainable Holy Grail or a Vital Breakthrough within Our Reach? *IEEE Security & Privacy*, 12(2):67–69, 2014.

[13] MagicDraw. `http://www.magicdraw.com`, last visited May 2013.

[14] Douglas E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing As a Discipline. *Communications of the ACM*, 32(1):9–23, 1989.

[15] Ida Solheim and Ketil Stølen. Technology Research Explained. Technical report, SINTEF ICT, A313, 2007.

[16] Joseph E. McGrath. *Groups: Interaction and Performance*. Prentice-Hall, 1983.

[17] The SecFutur project: Design of Secure and Energy-efficient Embedded Systems for Future Internet Application. `http://www.secfutur.eu`.

[18] James K. Peckol. *Embedded Systems: A Contemporary Design Tool*. John Wiley & Sons, 2007.

[19] Alan M. Davis. *Software Requirements: Analysis and Specification*. Prentice Hall Press, 1990.

[20] German Ministry of Defense. V Model: Software Lifecycle Process Model, Geenral Reprint No 250., 1992.

[21] Barry W. Boehm. *A Spiral Model of Software Development and Enhancement*. IEEE Computer Society Press, 1988.

[22] Bruce Powel Douglass. *Real Time UML Workshop for Embedded Systems*. Elsevier, 2007.

[23] Felice Balarin, Yosinori Watanabe, Harry Hsieh, Luciano Lavagno, Claudio Passerone, and Alberto Sangiovanni-Vincentelli. Metropolis: An Integrated Electronic System Design Environment. *Computer, IEEE*, 36(4):45–52, 2003.

[24] Ludovic Apvrille, Waseem Muhammad, Rabéa Ameur-Boulifa, Sophie Coudert, and Renaud Pacalet. A UML-based Environment for System Design Space Exploration. In *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1272–1275, 2006.

[25] Petru Eles. Lecture notes: Real-time and embedded systems. Technical report, Linköping university, Department of Computer and Information Science (IDA), 2012.

[26] Peter Marwedel. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. Springer Netherlands, 2011.

[27] Bran Selic. The Pragmatics of Model-driven Development. *IEEE Computer Society Press*, 20:19–25, 2003.

[28] Object Management Group. Unified Modeling Language: Superstructure, version 2.4.1. Document number: formal/2011-08-06, 2011.

[29] Jean Bézivin. On the Unification Power of Models. *Software and System Modeling*, 4:171–188, 2005.

[30] Object Management Group. `http://www.omg.org`, last visited July 2013.

[31] Object Management Group. Meta-Object Facilities: Core Specification, version 2.0. Document number: formal/06-01-01, 2001.

[32] Dragan Gasevic, Dragan Djuric, and Vladan Devedzic. *Model Driven Engineering and Ontology Development*. Springer-Verlag Berlin Heidelberg, 2009.

[33] Krzysztof Czarnecki and Simon Helsen. Feature-based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[34] Hans Vangheluwe. Invited Talk: Promises and Challenges of Model-Driven Engineering. In *European Conference on Software Maintenance and Reengineering (CSMR)*, pages 3 – 4. IEEE, 2011.

[35] Holger Giese, Tihamér Levendovszky, and Hans Vangheluwe. Summary of the Workshop on Multi-Paradigm Modeling: Concepts and Tool. In *International Conference on Models in Software Engineering (MoDELS)*, pages 252 – 262. Springer Berlin Heidelberg, 2007.

[36] Steven Kelly and Juha-Pekka Tolvannen. *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons, 2008.

[37] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. Assessing the State-of-Practice of Model-Based Engineering in the Embedded Systems Domain. In *International Conference on Model-Driven Engineering Languages and Systems (MoDELS)*, pages 166 – 182. Springer International Publishing, 2014.

[38] Florian Noyrit, Sébastien Gérard, François Terrier, and Bran Selic. Consistent Modeling Using Multiple UML Profiles. In *International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, pages 392 – 406. Springer Berlin Heidelberg, 2010.

[39] François Lagarde, Huáscar Espinoza, François Terrier, and Sébastien Gérard. Improving UML Profile Design Practices by Leveraging Conceptual Domain Models. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 445–448, 2007.

[40] Bran Selic. A Systematic Approach to Domain-specific Language Design Using UML. In *Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 2 – 9. IEEE Computer Society, 2007.

[41] Object Management Group. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, version 1.1. Document number: formal/2011-06-02, 2011.

[42] Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann. Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services. *Journal of Systems and Software, Elsevier*, 82(12):2068 – 2080, 2009.

[43] Frank Alexander Kraemer and Peter Herrmann. Automated Encapsulation of UML Activities for Incremental Development and Verification. In *International Conference on Model Driven Engineering, Languages and Systems (MoDELS)*, volume 5795, pages 571–585. Springer-Verlag, 2009.

[44] Frank Alexander Kraemer and Peter Herrmann. Reactive Semantics for Distributed UML Activities. In *Formal Techniques for Distributed Systems (FMOODS/FORTE)*, pages 17 – 31. Springer Berlin Heidelberg, 2010.

[45] Frank Alexander Kraemer and Peter Herrmann. Formalising Collaboration-oriented Service Specifications Using Temporal Logic. In *Networking and Electronic Commerce Research Conference*, 2007.

[46] Linda Ariani Gunawan, Peter Herrmann, and Frank Alexander Kraemer. Towards the Integration of Security Aspects into System Development using Collaboration-Oriented Models. In *International Conference on Security Technology (SecTech)*, volume 58, pages 72 – 85. Springer Berlin Heidelberg, 2009.

[47] Linda Ariani Gunawan, Frank Alexander Kraemer, and Peter Herrmann. A Tool-Supported Method for the Design and Implementation of Secure Distributed Applications. In *Engineering Secure Software and Systems (ESSoS)*, pages 142 – 155. Springer Berlin Heidelberg, 2011.

[48] Linda Ariani Gunawan and Peter Herrmann. Compositional Verification of Application-Level Security Properties. In *Engineering Secure Software and Systems (ESSoS)*, pages 75 – 90. Springer Berlin Heidelberg, 2013.

[49] Eclipse Modeling Framework Project (EMF). `http://www.eclipse.org/modeling/emf/`, last visited July 2013.

[50] Kermeta. `http://www.kermeta.org`, last visited July 2013.

[51] Frédéric Jouault and Jean Bézivin. KM3: a DSL for Metamodel Specification. In *IFIP WG 6.1 international conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, pages 171–185. Springer Berlin Heidelberg, 2006.

[52] Graphiti – a Graphical Tooling Infrastructure. `http://www.eclipse.org/graphiti/`, last visited July 2013.

[53] Graphical Modeling Project (GMP). `http://www.eclipse.org/modeling/gmp/`, last visited July 2013.

[54] Xtext. `http://www.eclipse.org/Xtext/`, last visited July 2013.

[55] Enterprise Architecture. `http://www.sparxsystems.com/products/ea/index.html`, last visited July 2013.

[56] Rhapsody. `http://www-03.ibm.com/software/products/us/en/ratirhapfami/`, last visited July 2013.

[57] MDT-UML2Tools. `http://www.wiki.eclipse.org/MDT-UML2Tools`, last visited July 2013.

[58] Papyrus project. `http://www.papyrusuml.org`, last visited July 2013.

[59] Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT). `http://www.omg.org/spec/QVT/`, last visited July 2013.

[60] ATL. `http://www.eclipse.org/atl/`, last visited July 2013.

[61] Henshin. `http://www.eclipse.org/henshin/`, last visited July 2013.

[62] EMorF. `http://www.emorf.org`, last visited July 2013.

[63] B. Chandrasekaran, John R. Josephson, and V. Richard Benjamins. What are Ontologies and Why Do We Need Them? In *Intelligent Systems*, volume 14, pages 20–26. IEEE Educational Activities Department, 1999.

[64] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, `http://www.w3.org/TR/owl2-syntax/`, last visited January 2013.

[65] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`, last visited January 2013.

[66] Protégé Editor. `http://www.protege.stanford.edu`, last visited March 2013.

[67] The OWL API. `http://www.owlapi.sourceforge.net`, last visited March 2013.

[68] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. *Semantic web, IOS Press*, 2(1):11 – 21, 2011.

[69] SPARQL in Protégé-OWL. `http://www.protege.stanford.edu/doc/sparql/`, last visited March 2013.

[70] Apache Jena Project. `http://www.jena.apache.org/`, last visited February 2013.

[71] OWL Web Ontology Language. `http://www.w3.org/TR/owl-ref/`, last visited July 2013.

[72] Almut Herzog, Nahid Shahmehri, and Claudiu Duma. An Ontology of Information Security. *Journal of Techniques and Applications for Advanced Information Privacy and Security, IGI Global*, pages 278–301, 2007.

[73] Stefan Fenz and Andreas Ekelhart. Formalizing Information Security Knowledge. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 183–194. ACM, 2009.

[74] Anya Kim, Jim Luo, and Myong Kang. Security Ontology for Annotating Resources. In *International Conference on On the Move to Meaningful Internet Systems (OTM)*, pages 1483 – 1499. Springer Berlin Heidelberg, 2005.

[75] Maria Karyda, Theodoros Balopoulos, Lazaros Gymnopoulos, Spyros Kokolakis, Costas Lambrinoudakis, Stefanos Gritzalis, and Stelios Dritsas. An Ontology for Secure e-government Applications. In *International Conference on Availability, Reliability and Security (ARES)*, pages 1033–1037. IEEE Computer Society, 2006.

[76] Ronald A. Howard. *Dynamic Probabilistic Systems, Volume II: Semi-Markov and Decision Processes*. John Wiley & Sons, New York, 1971.

[77] Bill Whyte and John Harrison. State of Practice in Secure Software: Experts' Views on Best Ways Ahead. *Software Engineering for Secure Systems: Industrial and Research Perspectives, IGI Global*, 2011.

[78] Acceleo. `http://www.eclipse.org/acceleo/`, last visited February 2013.

[79] Aamer Nadeem and Younus Javed. A Performance Comparison of Data Encryption Algorithms. In *International Conference on Information and Communication Technologies (ICICT)*, pages 84 – 89. IEEE, 2005.

[80] R. Stephen Preissig. Data Encryption Standard (DES) Implementation on the TMS320C6000, Application Report, 2000.

[81] Massimiliano Raciti and Simin Nadjm-Tehrani. Embedded Cyber-Physical Anomaly Detection in Smart Meters. In *Conference on Critical Information Infrastructures Security (CRITIS)*, pages 34 – 45. Springer Berlin Heidelberg, 2012.

[82] Petri Selonen. A Review of UML Model Comparison Approaches. In *Nordic Workshop on Model Driven Engineering*, pages 37 – 51. IT University of Göteborg, 2007.

[83] Lars Bendix and Pär Emanuelsson. Diff and Merge Support for Model-based Development. In *Workshop on Comparison and versioning of software models (CVSM)*, pages 31 – 34. ACM, 2008.

[84] EMF Compare; `http//:www.eclipse.org/emf/compare/`, last visited April 2013.

[85] Geri Georg, Kyriakos Anastasakis, Behzad Bordbar, Siv Hilde Houmb, Indrakshi Ray, and Manachai Toahchoodee. Verification and Trade-Off Analysis of Security Properties in UML System Models. In *IEEE Transactions on Software Engineering*, volume 36, pages 338 – 356. 2010.

[86] OMAP3530. `http://www.ti.com/product/omap3530`, last visited February 2013.

[87] Ontology Language Manchester Syntax. `http://www.w3.org/TR/owl2-manchester-syntax/`, last visited May 2013.

[88] HermiT Reasoner. `http://www.hermit-reasoner.com`, last visited september 2013.

[89] Vinay Igure and Ronald Williams. Taxonomies of Attacks and Vulnerabilities in Computer Systems. *Communications Surveys Tutorials, IEEE*, 10(1):6 – 19, 2008.

[90] Anton V. Uzunov and Eduardo B. Fernandez. An Extensible Pattern-based Library and Taxonomy of Security Threats for Distributed Systems. *Computer Standards & Interfaces, Elsevier*, 36(4):734 – 747, 2014.

[91] Srivaths Ravi, Anand Raghunathan, and Srimat Chakradhar. Tamper Resistance Mechanisms for Secure Embedded Systems. In *International Conference on VLSI Design (VLSID)*, pages 605 – 611. IEEE, 2004.

[92] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[93] Jesús M. Hermida, María Teresa Romá-Ferri, Andrés Montoyo, and Manuel Palomar. Reusing UML Class Models to Generate OWL Ontologies - A Use Case in the Pharmacotherapeutic Domain. In *International Conference on Knowledge Engineering and Ontology Development (KEOD)*, pages 281–286. INSTICC Press, 2009.

[94] Zhuoming Xu, Yuyan Ni, Wenjie He, Lili Lin, and Qin Yan. Automatic Extraction of OWL Ontologies from UML Class Diagrams. *World Wide Web*, 2012.

[95] XSD Type System. `http://www.w3.org/`, last visited March 2013.

[96] Simple Part-whole Relations in OWL Ontologies. `http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/`, last visited March 2013.

[97] Object Management Group. Ontology Definition Metamodel, version 1.0. Document number: formal/2009-05-01, 2009.

[98] Texas Instruments. `http://www.ti.com`, last visited March 2013.

[99] Cryptography for C64x+-based Devices. `http://www.ti.com/tool/c64xpluscrypto`, last visited January 2013.

[100] Andy Seaborne. SPARQL 1.1 Property Paths, `http://www.w3.org/TR/2010/WD-sparql11-property-paths-20100126/`, last visited Jan. 2013.

[101] Ray Kamal. *Embedded Systems: Architecture, Programming and Design*. Tata McGraw-Hill, 2009.

[102] Henrik Dibowski and Klaus Kabitzsch. Ontology-Based Device Descriptions and Device Repository for Building Automation Devices. *EURASIP Journal of Embedded Systems, Hindawi Publishing Corp.*, pages 3:1 – 3:17, 2011.

[103] Michael Unterkalmsteiner, Tony Gorschek, A.K.M. Moinul Islam, Chow Kian Cheng, Rahadian Bayu Permadi, and Robert Feldt. Evaluation and Measurement of Software Process Improvement – A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 38(2):398 – 424, 2012.

[104] Nathan Baddoo and Tracy Hall. De-motivators for Software Process Improvement: an Analysis of Practitioners Views. *Journal of Systems and Software, Elsevier Science Inc.*, pages 23 – 33, 2003.

[105] Mahmood Niazi, David Wilson, Didar Zowghi, and Bernard Wong. A Model for the Implementation of Software Process Improvement: An Empirical Study. In *Conference on Product Focused Software Process Improvement (PROFES)*, pages 1 – 16. Springer Berlin Heidelberg, 2004.

[106] Mass Soldal Lund, Bjørnar Solhaug, and Ketil Stølen. *Model-Driven Risk Analysis: The CORAS Approach*. Springer Publishing Company, Incorporated, 2010.

[107] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, pages 11 – 33, 2004.

[108] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons, 2005.

[109] Jan Jürjens. *Secure System Development with UML*. Springer-Verlag Berlin Heidelberg, 2005.

[110] Philip Koopman. *Better Embedded System Software*. Drumnadrochit Education LLC, 2010.

[111] J. Efrim Boritz. Is Practitioners' Views on Core Concepts of Information Integrity. *International Journal of Accounting Information Systems, Elsevier*, 6(4):260 – 279, 2005.

[112] Donn B Parker. *Toward a New Framework for Information Security, The Computer Security Handbook*. New York, NY: John Wiley & Sons, 2002.

[113] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. Smart Grid – The New and Improved Power Grid: A Survey. In *IEEE Communications Surveys and Tutorials*, pages 944 – 980, 2012.

[114] Frances M. Cleveland. Cyber Security Issues for Advanced Metering Infrastructure (AMI). *IEEE Power and Energy Society General Meeting: Conversion and Delivery of Electrical Energy in the 21st Century*, pages 1 – 5, 2008.

[115] SP 800-27 Rev. A: Engineering Principles for Information Technology Security (A Baseline for Achieving Security). Technical report, National Institute of Standards and Technology (NIST), 2004.

[116] ISO/IEC 27000, Information technology – Security techniques – Information security management systems. Technical report, International Organization for Standardization (ISO), 2009.

[117] The Smart Grid Interoperability Panel – Cyber Security Working Group, Guidelines for Smart Grid Cyber Security: Vol. 1, Smart Grid Cyber Security Strategy, Architecture, and High-Level Requirements. Technical report, NIST Interagency or Internal Report, (NISTIR 7628).

[118] Dimitrios N. Serpanos and Artemios G. Voyiatzis. Security Challenges in Embedded Systems. *ACM Transactions on Embedded Computing Systems*, 12(1):66:1 – 66:10, 2013.

[119] Le Minh Sang Tran, Bjørnar Solhaug, and Ketil Stølen. An Approach to Select Cost-Effective Risk Countermeasures Exemplified in CORAS. In *Data and Applications Security and Privacy (DBSec)*, pages 266 – 273. Springer Berlin Heidelberg, 2013.

[120] Armen Der Kiureghian and Ove Ditlevsen. Aleatory or Epistemic? Does It Matter? *Journal of Structural Safety, Risk Acceptance and Risk Communication Risk Acceptance and Risk Communication, Elsevier*, 31(2):105 – 112, 2009.

[121] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. DAG-Based Attack and Defense Modeling: Don't Miss the Forest for the Attack Trees. *Computer Science Review, Elsevier*, abs/1303.7397, 2014.

[122] Ronald A. Howard. *Dynamic Probabilistic Systems. 2, Semi-Markov and Decision Processes*. John Wiley & Sons, 1971.

[123] Folker den Braber, Ida Hogganvik, Soldal Lund, Ketil Stølen, and Fredrik Vraalsen. Model-Based Security Analysis in Seven Steps – a Guided Tour to the CORAS Method. *BT Technology Journal, Kluwer Academic Publishers-Consultants Bureau*, 25(1):101–117, 2007.

[124] Youngja Park, Christopher Gates, and Stephen C. Gates. Estimating Asset Sensitivity by Profiling Users. In *European Symposium on Research in Computer Security (ESORICS)*, pages 94–110. Springer Berlin Heidelberg, 2013.

[125] Samir Ouchani, Otmane Mohamed, and Mourad Debbabi. A Formal Verification Framework for SysML Activity Diagrams. *Journal of Expert Systems with Applications, Elsevier*, 41(6):2713–2728, 2014.

[126] Gianfranco Ciardo, Reinhard German, and Christoph Lindemann. A Characterization of the Stochastic Process Underlying a Stochastic Petri Net. *IEEE Transactions on Software Engineering*, 20(7):506 – 515, 1994.

[127] Martin Erich Jobst. Security and Privacy in the Smart Energy Grid. In *Smart Grid Security Workshop (SmartGridSec)*. ACM, 2014.

[128] Wenye Wang and Zhuo Lu. Cyber Security in the Smart Grid: Survey and Challenges. *Computer Networks, Elsevier*, 57(5):1344 – 1371, 2013.

[129] Hyun Sang Cho, Tatsuya Yamazaki, and Minsoo Hahn. AERO: Extraction of User's Activities from Electric Power Consumption Data. *IEEE Transactions on Consumer Electronics*, 56(3):2011 – 2018, 2010.

[130] Mikhail A. Lisovich and Stephen B. Wicker. Privacy Concerns in Upcoming Residential and Commercial Demand-response Systems. In *IEEE Proceedings on Power Systems*, volume 1, pages 1 – 10, 2008.

[131] Florian Arnold, Holger Hermanns, Reza Pulungan, and Mariëlle Stoelinga. Time-dependent Analysis of Attacks. In *Conference on Principles of Security and Trust (POST)*, pages 285 – 305. Springer Berlin Heidelberg, 2014.

[132] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics, 2009.

[133] Gethin Norman, Marta Kwiatkowska, and David Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *International Conference on Computer Aided Verification (CAV)*, pages 585 – 591. Springer-Verlag, 2011.

[134] Maria Vasilevskaya and Simin Nadjm-Tehrani. Model-based Security Risk Analysis for Networked Embedded Systems. In *International Conference on Critical Information Infrastructures Security (CRITIS)*. Springer, 2014.

[135] Leyla Bilge and Tudor Dumitras. Before We Knew It: An Empirical Study of Zero-Day Attacks In The Real World. In *ACM Conference on Computer and Communications Security (CCS)*, pages 833 – 844, 2012.

[136] Omar H. Alhazmi and Yashwant K. Malaiya. Modeling the Vulnerability Discovery Process. In *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pages 129–138, 2005.

[137] Andy Ozment. Improving Vulnerability Discovery Models. In *ACM Workshop on Quality of Protection (QoP)*, pages 6 – 11, 2007.

[138] Erland Jonsson and Tomas Olovsson. A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior. *IEEE Transactions on Software Engineering*, 23(4):235 – 245, 1997.

[139] Mohamed Kaâniche, Yves Deswarte, Eric Alata, Marc Dacier, and Vincent Nicomette. Empirical Analysis and Statistical Modelling of

Attack Processes Based on Honeypots. *CoRR, http://www.arxiv.org/abs/0704.0861*, 2007.

[140] Jaafar Almasizadeh and Mohammad Abdollahi Azgomi. A Stochastic Model of Attack Process for the Evaluation of Security Metrics. *Journal of Computer Networks, Elsevier*, 57(10):2159 – 2180, 2013.

[141] Axel Thümmler, Peter Buchholz, and Miklos Telek. A Novel Approach for Phase-type Fitting with the EM Algorithm. *IEEE Transactions on Dependable Secure Computing*, 3(3):245 – 258, 2006.

[142] Teodor Sommestad, Hannes Holm, and Mathias Ekstedt. Estimates of Success Rates of Remote Arbitrary Code Execution Attacks. *Information Management & Computer Security*, 20(2):107–122, 2012.

[143] Teodor Sommestad, Hannes Holm, and Mathias Ekstedt. Estimates of Success Rates of Denial-of-Service Attacks. In *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 21 – 28, 2011.

[144] Teodor Sommestad, Hannes Holm, and Mathias Ekstedt. Quantifying the Effectiveness of Intrusion Detection Systems in Operation through Domain Experts. In *IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 21–28, 2011.

[145] Simon Parsons. Current Approaches to Handling Imperfect Information in Data and Knowledge Bases. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):353 – 372, 1996.

[146] Nan Feng and Jing Xie. A Bayesian Networks-based Security Risk Analysis Model for Information Systems Integrating the Observed Cases with Expert Experience. *Scientific Research and Essays*, 7(10):1103–1112, 2012.

[147] David M. Nicol, William H. Sanders, and Kishor S. Trivedi. Model-based Evaluation: from Dependability to Security. *IEEE Transactions on Dependable and Secure Computing*, pages 48 – 65, 2004.

[148] Aivo Jürgenson and Jan Willemson. Processing Multi-Parameter Attack Trees with Estimated Parameter Values. In *International Conference on Advances in Information and Computer Security (IWSEC)*, pages 308 – 319. Springer-Verlag, 2007.

[149] Bharat B. Madan, Kalyanaraman Vaidyanathan, and Kishor S. Trivedi. A Method for Modelling and Quantifying the Security Attributes of Intrusion Tolerant Systems. *Journal of Performance Evaluation, Elsevier*, 56(1 – 4):167 – 186, 2004.

[150] Fernando Herrera, Héctor Posadas, Pablo Peñil, Eugenio Villar, Francisco Ferrero, Raúl Valencia, and Gianluca Palermo. The COMPLEX Methodology for UML/MARTE Modeling and Design Space Exploration of Embedded Systems. *Journal of Systems Architecture, Elsevier*, 60(1):55 – 78, 2014.

[151] Murray Woodside, Dorina C. Petriu, Dorin B. Petriu, Hui Shen, Toqeer Israr, and Jose Merseguer. Performance by Unified Model Analysis (PUMA). In *ACM Workhsop on Software and Performance (WOSP)*, pages 1–12. ACM, 2005.

[152] Thomas Robert and Vincent Perrier. CoFluent Methodology for UML. *A CoFluent Design, White Paper*, 2010.

[153] Marcio F. da S. Oliveria, Lisane B. de Brisolara, Luigi Carro, and Flavio R. Wagner. Early Embedded Software Design Space Exploration Using UML-Based Estimation. In *International Workshop on Rapid System Prototyping*, pages 24 – 32. IEEE Computer Society, 2006.

[154] Federico Ciccozzi, Antonio Cicchetti, and Mikael Sjödin. Round-trip Support for Extra-functional Property Management in Model-driven Engineering of Embedded Systems. *Information and Software Technology, Butterworth-Heinemann*, 55(6):1085 – 1100, 2012.

[155] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[156] Ivica Crnkovic. Component-based Approach for Embedded Systems. In *International Conference on Software Engineering (ICSE)*, pages 712–713. IEEE, 2005.

[157] Ed Brinksma, Geoff Coulson, Ivica Crnkovic, Andy Evans, Sébastien Gérard, Susanne Graf, Holger Hermanns, Bengt Jonsson, Anders Ravn, Philippe Schnoebelen, François Terrier, Angelika Votintseva, and Jean-Marc Jézéquel. Component-based Design and Integration Platforms. Technical report, Project IST-2001-34820. ARTIST, 2003.

[158] Ivica Crnkovic, Séverine Sentilles, Aneta Vulgarakis, and Michel Chaudron. A Classification Framework for Software Component Models. *IEEE Transaction on Software Engineering*, 37(5):593 – 615, 2011.

[159] Ivica Crnkovic. *Building Reliable Component-Based Software Systems*. Artech House, Inc., 2002.

[160] Ivica Crnkovic. Managing Complexity and Predictability in Embedded Systems: Applying Component-based Development. In *International Workshop on Software Engineering for Embedded Systems (SEES)*, pages 1–1. IEEE Press, 2012.

[161] Alberto Sangiovanni-Vincentelli and Marco Di Natale. Embedded System Design for Automotive Applications. *IEEE Computer Society Press*, 40(10):42–51, 2007.

[162] Hugh Maaskant. *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*, chapter A Robust Component Model for Consumer Electronic Products, pages 167 – 192. Springer Netherlands, 2005.

[163] Séverine Sentilles, Aneta Vulgarakis, Tomáš Bureš, Jan Carlson, and Ivica Crnković. A Component Model for Control-Intensive Distributed Embedded Systems. In *International Symposium on Component-Based Software Engineering (CBSE)*, pages 310 – 317. Springer Berlin Heidelberg, 2008.

[164] Petr Hošek, Tomáš Pop, Tomáš Bureš, Petr Hnětynka, and Michal Malohlava. Comparison of Component Frameworks for Real-Time Embedded Systems. In *International Conference on Component-Based Software Engineering (CBSE)*, pages 21–36. Springer-Verlag Berlin, Heidelberg, 2010.

[165] Tomáš Pop, Petr Hnětynka, Petr Hošek, Michal Malohlava, and Tomáš Bureš. Comparison of Component Frameworks for Real-time Embedded Systems. *Journal of Knowledge and Information Systems, Springer London*, 40(1):127 – 170, 2014.

[166] Edsger W. Dijkstra. On the Role of Scientific Thought. *Selected Writings on Computing: A Personal Perspective, Springer-Verlag*, pages 60 – 66, 1982.

[167] Robert France, Indrakshi Ray, Geri Georg, and Sudipto Ghosh. Aspect-Oriented Approach to Early Design Modelling. *IEEE Proceedings Software*, 151(4):173 – 185, 2004.

[168] Manuel Wimmer, Andrea Schauerhuber, Gerti Kappel, Werner Retschitzegger, Wieland Schwinger, and Elizabeth Kapsammer. A Survey on UML-based Aspect-oriented Design Modeling. *ACM Computing Surveys*, 43(4):28:1 – 28:33, 2011.

[169] Makan Pourzandi, Djedjiga Mouheb, Chamseddine Talhi, Vitor Lima, Mourad Debbabi, and Lingyu Wang. Weaving Security Aspects into UML 2.0 Design Models. In *Workshop on Aspect Oriented Modeling (AOM)*, pages 7 – 12. ACM, 2009.

[170] Marco A. Wehrmeister and Gian R. Berkenbrock. AMoDE-RT: Advancing Model-Driven Engineering for Embedded Real-Time Systems. In *IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, pages 1 – 7, 2013.

[171] Marco A. Wehrmeister, Carlos Eduardo Pereira, and Franz J. Rammig. Aspect-Oriented Model-Driven Engineering for Embedded Systems Applied to Automation Systems. *IEEE Transactions on Industrial Informatics*, 9(4):2373–2386, 2013.

[172] Lichen Zhang. Aspect-oriented Modeling of Railway Cyber Physical Systems Based on the Extension of AADL. In *IEEE International Conference on High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing*, pages 2104 – 2111, 2013.

[173] Éric Piel, Samy Meftali, Jean luc Dekeyser, Rabie Ben Atitallah, Smaïl Niar, Anne Etien, and Pierre Boulet. Gaspard2: from MARTE to SystemC Simulation. In *Workshop on Modeling and Analysis of Real-Time and Embedded Systems (at DATE)*, 2008.

[174] Object Management Group. Systems Modeling Language: version 1.3. Document number: formal/2012-06-01, 2012.

[175] Murray Woodside, Dorina C. Petriu, Dorin B. Petriu, Jing Xu, Tauseef Israr, Geri Georg, Robert France, James M. Bieman, Siv Hilde Houmb, and Jan Jürjens. Performance Analysis of Security Aspects by Weaving Scenarios Extracted from UML models. *Journal of Systems and Software, Elsevier Science Inc.*, 82(1):56 – 74, 2009.

[176] Marco A. Wehrmeister, Edison P. Freitas, Carlos E. Pereira, and Franz Rammig. GenERTiCA: A Tool for Code Generation and Aspects Weaving. In *IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 234 – 238, 2008.

[177] Egor Bondarev, Michel Chaudron, and Peter H. N. de With. CARAT: a Toolkit for Design and Performance Analysis of Component-Based Embedded Systems. In *Conference on Design, Automation and Test in Europe (DATE)*, pages 1024 – 1029. EDA Consortium, 2007.

[178] ARTEMIS-JU-216682, CHESS. `http://www.chess-project.ning.com/`, last visited March 2013.

[179] Andrey Chechulin, Vasily Desnitsky, and Igor Kotenko. Configuration-based Approach to Embedded Device Security. In *International Conference on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS)*, pages 270 – 285. Springer Berlin Heidelberg, 2012.

[180] Igor Kotenko and Vasily Desnitsky. Expert Knowledge Based Design and Verification of Secure Systems with Embedded Devices. In *IFIP WG 8.4, 8.9, TC 5 International Cross-Domain Conference (CD-ARES)*, pages 194 – 210. Springer International Publishing, 2014.

[181] Phil Tetlow, Jeff Z. Pan, Daniel Oberle, Evan Wallace, Michael Uschold, and Elisa Kendall. Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering. `http://www.w3.org/2001/sw/BestPractices/SE/ODA/`, last visited may 2013.

[182] Tobias Walter, Fernando Silva Parreiras, and Steffen Staab. An Ontology-based Framework for Domain-specific Modeling. In *Software & Systems Modeling*, volume 13, pages 83 – 108. Springer Berlin Heidelberg, 2014.

[183] Jianjun Yi, Ying Cheng, and Chunhua Gu. A Reconfigurable Design Method of Embedded Control System based on Ontology. In *International Conference on E-Product E-Service and E-Entertainment (ICEEE)*, pages 1 – 4. IEEE, 2010.

[184] Dennis Wagelaar. *Platform Ontologies for the Model-Driven Architecture*. PhD thesis, Vrije Universiteit Brussel, Department of Computer Science, 2010.

[185] Bedir Tekinerdoğan, Sevcan Bilir, and Cem Abatlevi. Integrating Platform Selection Rules in the Model Driven Architecture Approach. In *European Conference on Model Driven Architecture: Foundations and Applications (MDAFA)*, pages 159–173. Springer-Verlag, 2003.

[186] Carlos Blanco, Joaquin Lasheras, Rafael Valencia-García, Eduardo Fernández-Medina, Ambrosio Toval, and Mario Piattini. A Systematic Review and Comparison of Security Ontologies. In *International Conference on Availability, Reliability and Security (ARES)*, pages 813 – 820. IEEE, 2008.

[187] Amina Souag, Camille Salinesi, and Isabelle Comyn-Wattiau. Ontologies for Security Requirements: A Literature Survey and Classification. In *Advanced Information Systems Engineering Workshops (CAiSE)*, volume 112, pages 61 – 69. Springer Berlin Heidelberg, 2012.

[188] Stefan Gärtner, Thomas Ruhroth, Kurt Schneider, and Jan Jürjens. Maintaining Requirements for Long-Living Software Systems by Incorporating Security Knowledge. In *IEEE International Requirements Engineering Conference (RE)*, pages 103 – 112, 2014.

[189] Amina Souag, Raul Mazo, Camille Salinesi, and Isabelle Comyn-Wattiau. A Security Ontology for Security Requirements Elicitation. In *International Symposium Engineering Secure Software and Systems (ESSoS)*, volume 8978, pages 157 – 177. Springer International Publishing, 2015.

[190] Thomas Ruhroth, Stefan Gärtner, Jens Bürger, Jan Jürjens, and Kurt Schneider. Towards Adaptation and Evolution of Domain-Specific

Knowledge for Maintaining Secure Systems. In *International Conference Product-Focused Software Process Improvement(PROFES)*, volume 8892, pages 239–253. Springer International Publishing, 2014.

[191] Muhammad Javed, Yalemisew M. Abgaz, and Claus Pahl. Ontology Change Management and Identification of Change Patterns. *Journal of Data Semantics, Springer-Verlag*, 2(2):119 – 143, 2013.

[192] Thomas Ruhroth and Jan Jürjens. Supporting Security Assurance in the Context of Evolution: Modular Modeling and Analysis with UMLsec. In *IEEE International Symposium on High-Assurance Systems Engineering (HASE)*, pages 177 – 184, 2012.

[193] Denis Hatebur, Maritta Heisel, Jan Jürjens, and Holger Schmidt. Systematic Development of UMLsec Design Models Based on Security Requirements, in: Fundamental Approaches to Software Engineering. In *International Conference on Fundamental Approaches to Software Engineering: Part of the Joint European Conferences on Theory and Practice of Software (FASE/ETAPS), Springer-Verlag*, pages 232–246. Springer-Verlag, 2011.

[194] Siv Hilde Houmb, Shareeful Islam, Eric Knauss, Jan Jürjens, and Kurt Schneider. Eliciting Security Requirements and Tracing Them to Design: an Integration of Common Criteria, Heuristics, and UMLsec. *Journal of Requirements Engineering, Springer-Verlag New York, Inc.*, 15(1):63 – 93, 2010.

[195] *Common Criteria for Information Technology Security Evaluation, Version 3.1., Revision 4, CCMB-2012-09-001*, 2012.

[196] Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *International Conference on The Unified Modeling Language (UML)*, pages 426–441. Springer-Verlag, 2002.

[197] David Basin, Manuel Clavel, and Marina Egea. A Decade of Model-Driven Security. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 1 – 10, 2011.

[198] Geri Georg, Indrakshi Ray, Kyriakos Anastasakis, Behzad Bordbar, Manachai Toahchoodee, and Siv Hilde Houmb. An Aspect-Oriented Methodology for Designing Secure Applications. *Information and Software Technology, Elsevier*, 51(5):846 – 864, 2009.

[199] Siv Hilde Houmb, Geri Georg, Dorina Petriu, Behzad Bordbar, Indrakshi Ray, Kyriakos Anastasakis, and Robert France. Balancing Security and Performance Properties During System Architectural Design. *Software Engineering for Secure Systems: Industrial and Research Perspectives, IGI Global*, pages 155 – 191, 2011.

[200] Eduardo Fernandez-Buglioni. *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. Wiley Publishing, 2013.

[201] Holger Schmidt, Denis Hatebur, and Maritta Heisel. A Pattern-Based Method to Develop Secure Software. *Software Engineering for Secure Systems: Industrial and Research Perspectives, IGI Global*, pages 32 – 74, 2011.

[202] Anton V. Uzunov, Eduardo B. Fernandez, and Katrina Falkner. Securing Distributed Systems Using Patterns: A Survey. *Computers & Security, Elsevier*, 31(5):681 – 703, 2012.

[203] Óscar Sánchez Ramón, Fernando Molina, Jesús García Molina, José Ambrosio, and Toval Álvarez. ModelSec: A Generative Architecture for Model-Driven Security. *Journal of Universal Computer Science*, 15(15):2957 – 2980, 2009.

[204] Brahim Hamid, Sigrid Gürgens, Christophe Jouvray, and Nicolas Desnos. Enforcing S&D Pattern Design in RCES with Modeling and Formal Approaches. In *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, pages 319–333, 2011.

[205] Gabriel Pedroza, Ludovic Apvrille, and Daniel Knorreck. AVATAR: A SysML Environment for the Formal Verification of Safety and Security Properties. In *IEEE International Conference on New Technologies of Distributed Systems (NOTERE)*, pages 1 – 10, 2011.

[206] Michael Hafner, Mukhtiar Memon, and Muhammad Alam. Modeling and Enforcing Advanced Access Control Policies in Healthcare Systems with SECTET. In *Models in Software Engineering*, pages 132 – 144. Springer-Verlag, 2008.

[207] Anton V. Uzunov, Eduardo B. Fernandez, and Katrina Falkner. Engineering Security into Distributed Systems: A Survey of Methodologies. *Journal of Universal Computer Science*, 18(20):2920 – 3006, 2012.

[208] Kresimir Kasal, Johannes Heurix, and Thomas Neubauer. Model-Driven Development Meets Security: An Evaluation of Current Approaches. In *Hawaii International Conference on System Sciences (HICSS), IEEE*, pages 1–9, 2011.

[209] Phu H. Nguyen, Jacques Klein, Yves Le Traon, and Max E. Kramer. A Systematic Review of Model Driven Security. In *Asia-Pacific Software Engineering Conference (APSEC)*, volume 1, pages 432 – 441. IEEE, 2013.

[210] Jostein Jensen and Martin Gilje Jaatun Security in Model Driven Development: A Survey. In *International Conference on Availability, Reliability and Security (ARES), IEEE*, pages 704–709. IEEE, 2011.

[211] Levi Lucio, Qin Zhang, Phu H. Nguyen, Moussa Amrani, Jacques Klein, Hans Vangheluwe, and Yves Le Traonb. Advances in Model-driven Security. *Advances in Computers, Elsevier*, 93:103 – 152, 2014.

[212] David Basin, Manuel Clavel, Jürgen Doser, and Marina Egea. Automated Analysis of Security-design Models. *Journal of Information and Software Technology, Butterworth-Heinemann*, 51(5):815–831, 2009.

[213] Olawande Daramola, Guttorm Sindre, and Thomas Moser. Ontology-Based Support for Security Requirements Specification Process. In *Workshops On the Move to Meaningful Internet Systems (OTM)*, pages 194–206. Springer Berlin Heidelberg, 2012.

[214] Wentao Kang and Ying Liang. A Security Ontology with MDA for Software Development. In *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CYBERC)*, pages 67–74. IEEE Computer Society, 2013.

[215] Stelios Dritsas, Lazaros Gymnopoulos, Maria Karyda, Theodoros Balopoulos, Spyros Kokolakis, Costas Lambrinoudakis, and Stefanos Gritzalis. Employing Ontologies for the Development of Security Critical Applications: The Secure e-poll Paradigm. In *IFIP Conference e-Commerce, e-Business, and e-Government (I3E)*, pages 187–201. Springer US, 2005.

[216] Eduardo B. Fernandez, Hironori Washizaki, Nobukazu Yoshioka, Atsuto Kubo, and Yoshiaki Fukazawa. Classifying Security Patterns. In *Asia-Pacific Web Conference on Progress in WWW Research and Development (APWeb)*, volume 4976, pages 342 – 347. Springer Berlin Heidelberg, 2008.

[217] Michael VanHilst, Eduardo B. Fernández, and Fabrício A. Braz. A Multi-Dimensional Classification for Users of Security Patterns. *Journal of Research and Practice in Information Technology, Australian Computer Society Inc.*, 41(2), 2009.

[218] Munawar Hafiz, Paul Adamczyk, and Ralph E. Johnson. Organizing Security Patterns. In 4, editor, *Software, IEEE*, volume 24, pages 52–60, 2007.

[219] Hironori Washizaki, Eduardo B. Fernandez, Katsuhisa Maruyama, Atsuto Kubo, and Nobukazu Yoshioka. Improving the Classification of Security Patterns. In *International Workshop on Database and Expert Systems Application (DEXA)*, pages 165 – 170. IEEE, 2009.

[220] Phu H. Nguyen, Jacques Klein, and Yves Le Traon. Model-Driven Security with A System of Aspect-Oriented Security Design Patterns. In *Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling (VAO)*, pages 51:51–51:54. ACM, 2014.

[221] Jose Fran. Ruiz, Rajesh Harjani, Antonio Mana, Vasily Desnitsky, Igor Kotenko, and Andrey Chechulin. A Methodology for the Analysis and Modeling of Security Threats and Attacks for Systems of Embedded Components. In *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 261 – 268. IEEE Computer Society, 2012.

[222] Jose Fran. Ruiz, Marcos Arjona, Antonio Mana, and Niklas Carstens. Secure Engineering and Modelling of a Metering Devices System. In *International Conference on Availability, Reliability and Security (ARES)*, pages 418–427. IEEE, 2013.

[223] Yomna Ali, Sherif El-Kassas, and Mohy Mahmoud. A Rigorous Methodology for Security Architecture Modeling and Verification. In *International Conference on System Sciences (HICSS)*, pages 1 – 10. IEEE, 2009.

[224] Brahim Hamid, Nicolas Desnos, Cyril Grepet, and Christophe Jouvray. Model-Based Security and Dependability Patterns in RCES – the TERESA Approach. In *International Workshop on Security and Dependability for Resource Constrained Embedded Systems (S&D4RCES)*. ACM, 2010.

[225] Matthew Eby, Jan Werner, Gabor Karsai, and Akos Ledeczi. Integrating Security Modeling into Embedded System Design. *IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS)*, pages 221 – 228, March 2007.

[226] Mehrdad Saadatmand and Thomas Leveque. Modeling Security Aspects in Distributed Real-Time Component-Based Embedded Systems. In *International Conference on Information Technology : New Generations (ITNG)*, pages 437 – 444. IEEE, 2012.

[227] Ludovic Apvrille and Yves Roudier. SysML-Sec: A Model-driven Environment for Developing Secure Embedded Systems. In *8ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR-SSI)*, Mont-de-Marsan, France, 2013.

[228] Ludovic Apvrille and Yves Roudier. SysML-Sec – A Model Driven Approach for Designing Safe and Secure Systems. In *International Conference on Model-Driven Engineering and Software Development (Modelsward)*. SCITEPRESS Digital Library, 2015.

[229] Seraj Fayyad and Josef Noll. Security and Safety Composition Methodology. In *International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services (CENTRIC)*, 2014.

[230] B. Barber and J. Davey. The Use of the CCTA Risk Analysis and Management Methodology CRAMM in Health Information Systems. In *International Congress on Medical Informatics (MEDINFO)*, pages 1589 – 1593, 1992.

[231] Bilge Karabacak and Ibrahim Sogukpinar. ISRAM: Information Security Risk Analysis Method. *Computers & Security, Elsevier*, 24(2):147 – 159, 2005.

[232] Alberts Christopher, Sandra Behrens, Richard Pethia, and William Wilson. Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) Framework, Version 1.0. Technical report, Technical Report CMU/SEI-99-TR-017. ESC-TR-99-017, Carnegie Mellon. Software Engineering Institute, 1999.

[233] Wayne Boyer and Miles McQueen. Ideal Based Cyber Security Technical Metrics for Control Systems. In *International Conference on Critical Information Infrastructures Security (CRITIS)*, pages 246 – 260. Springer-Verlag, 2007.

[234] Gary Stoneburner, Alice Y. Goguen, and Alexis Feringa. SP 800-30: Risk Management Guide for Information Technology Systems. Technical report, National Institute of Standards & Technology (NIST), 2002.

[235] IEC 31010 – Risk Management – Risk Assessment Techniques. Technical report, International Organization for Standardization (ISO), 2009.

[236] DHS Risk Steering Committee. DHS Risk Lexicon, 2010.

[237] Francesco Flammini, Stefano Marrone, Nicola Mazzocca, and Valeria Vittorini. Petri Net Modelling of Physical Vulnerability. In *Workshop Critical Information Infrastructure Security (CRITIS)*, volume 6983, pages 128 – 139. Springer Berlin Heidelberg, 2013.

[238] Geri Georg, Kyriakos Anastasakis, Behzad Bordbar, Siv Hilde Houmb, Indrakshi Ray, and Manachai Toahchoodee. Verification and Trade-off Analysis of Security Properties in UML System Models. *IEEE Transactions on Software Engineering*, 36(3):338–356, 2010.

[239] Teodor Sommestad, Mathias Ekstedt, and Pontus Johnson. A Probabilistic Relational Model for Security Risk Analysis. *Computers & Security, Elsevier*, 29(6):659 – 679, 2010.

[240] Marco D. Aime, Andrea Atzeni, and Paolo C. Pomi. AMBRA: Automated Model-based Risk Analysis. *ACM Workshop on Quality of Protection*, pages 43 – 48, 2007.

[241] Vilhelm Verendel. Quantified Security is a Weak Hypothesis: A Critical Survey of Results and Assumptions. In *New Security Paradigms Workshop (NSPW)*, pages 37 – 50. ACM, 2009.

[242] Sardar Muhammad Sulaman, Kim Weyns, and Martin Höst. A Review of Research on Risk Analysis Methods for IT Systems. In *International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 86 – 96. ACM, 2013.

[243] Manuel Rudolph and Reinhard Schwarz. A Critical Survey of Security Indicator Approaches. In *International Conference on Availability, Reliability and Security (ARES)*, pages 291 – 300. IEEE, 2012.

[244] Leanid Krautsevich, Fabio Martinelli, and Artsiom Yautsiukhin. Formal Approach to Security Metrics.: What Does "More Secure" Mean for You? In *European Conference on Software Architecture (ECSA)*, pages 162 – 169. ACM, 2010.

[245] Wayne Jansen. *Directions in Security Metrics Research*. National Institute of Standards & Technology (NIST), 2009.

[246] Binbin Chen, David M. Nicol, William H. Sanders, Rui Tan, William G. Temple, Nils Ole Tippenhauer, An Hoa Vu, and David K. Y. Yau. Go with the Flow: Toward Workflow-Oriented Security Assessment. In *New Security Paradigms Workshop (NSPW)*, pages 65 – 76. ACM, 2013.

[247] Nils Ole Tippenhauer, William G. Temple, An Hoa Vu, Binbin Chen, David M. Nicol, Zbigniew Kalbarczyk, and William H. Sanders. Automatic Generation of Security Argument Graphs. *CoRR*, abs/1405.7475.

[248] An Hoa Vu, Nils Ole Tippenhauer, Binbin Chen, David M. Nicol, and Zbigniew Kalbarczyk. CyberSAGE: A Tool for Automatic Security Assessment of Cyber-Physical Systems. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 384–387. Springer International Publishing, 2014.

[249] Jonathan D. Weiss. A System Security Engineering Process. In *National Computer Security Conference*, pages 572 – 581. National Institute of Standards and Technology/National Computer Security Center, 1991.

[250] Chris Salter, O. Sami Saydjari, Bruce Schneier, and Jim Wallner. Toward a Secure System Engineering Methodology. In *Workshop on New Security Paradigms (NSPW)*, pages 2 – 10. ACM, 1998.

[251] Bruce Schneier. Attack Trees: Modeling Security Threats. In *Dr. Dobb's Journal*, volume 24, pages 21 – 29, 1999.

[252] Roberto Vigo, Alessandro Bruni, and Ender Yüksel. Security Games for Cyber-Physical Systems. In *Nordic Conference on Secure IT Systems (NordSec)*, volume 8208, pages 17 – 32. Springer Berlin Heidelberg, 2013.

[253] Bharat B. Madan, Katerina Goševa-Popstojanova, Kalyanaraman Vaidyanathan, and Kishor S. Trivedi. A Method for Modeling and Quantifying the Security Attributes of Intrusion Tolerant Systems. *Journal of Performance Evaluation, Elsevier*, 56(1 – 4):167 – 186, 2004.

[254] Ahto Buldas and Aleksandr Lenin. New Efficient Utility Upper Bounds for the Fully Adaptive Model of Attack Trees. In *International Conference on Decision and Game Theory for Security (GameSec)*, volume 8252, pages 192 – 205. Springer International Publishing, 2013.

[255] Aivo Jürgenson and Jan Willemson. On Fast and Approximate Attack Tree Computations. In *International Conference Information Security, Practice and Experience (ISPEC)*, volume 6047, pages 56 – 66. Springer Berlin Heidelberg, 2010.

[256] Aivo Jürgenson and Jan Willemson. Computing Exact Outcomes of Multi-parameter Attack Trees. In *Confederated International On the Move to Meaningful Internet Systems (OTM)*, volume 5332, pages 1036 – 1051. Springer Berlin Heidelberg, 2008.

[257] Cynthia Phillips and Laura Painton Swiler. A Graph-based System for Network-Vulnerability Analysis. In *Workshop on New Security Paradigms (NSPW)*, pages 71 – 79. ACM, 1998.

[258] Elizabeth LeMay, Michael D. Ford, Ken Keefe, William H. Sanders, and Carol Muehrcke. Model-based Security Metrics Using ADversary View Security Evaluation (ADVISE). In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 191 – 200. ACM, 2011.

[259] Wolter Pieters and Mohsen Davarynejad. Calculating Adversarial Risk from Attack Trees: Control Strength and Probabilistic Attackers. In *International Workshop on Quantitative Aspects in Security Assurance (QASA)*, pages 201–215. Springer International Publishing, 2014.

[260] Michael D. Ford, Ken Keefe, Elizabeth LeMay, William H. Sanders, and Carol Muehrcke. Implementing the ADVISE security modeling formalism in Möbius. In *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1 – 8, 2013.

[261] Andrea S. Atzeni, Cesare Cameroni, Shamal Faily, John Lyle, and Ivan Flechais. Here's Johnny: A Methodology for Developing Attacker Personas. In *International Conference on Availability, Reliability and Security (ARES)*, pages 722 – 727. IEEE Computer Society, 2011.

[262] Arpan Roy, Dong Seong Kim, and Kishor S. Trivedi. ACT: Attack Countermeasure Trees for Information Assurance Analysis. In *INFOCOM IEEE Conference on Computer Communications Workshops*, pages 1 – 2, 2010.

[263] Arpan Roy, Dong Seong Kim, and Kishor S. Trivedi. Cyber Security Analysis Using Attack Countermeasure Trees. In *Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW)*, number 28:1 – 28:4. ACM, 2010.

[264] Stefano Bistarelli, Fabio Fioravanti, and Pamela Peretti. Defense Trees for Economic Evaluation of Security Investments. In *International Conference on Availability, Reliability and Security (ARES)*. IEEE, 2006.

[265] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Foundations of Attack-defense Trees. In *International Conference on Formal Aspects of Security and Trust (FAST)*, pages 80–95. Springer-Verlag, 2011.

[266] Zaruhi Aslanyan and Flemming Nielson. Pareto Efficient Solutions of Attack-Defence Trees. In *International Conference on Principles of Security and Trust (POST)*, pages 95–114. Springer Berlin Heidelberg, 2015.

[267] Frank Alexander Kraemer and Peter Herrmann. Transforming Collaborative Service Specifications into Efficiently Executable State Machines. In *International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT)*, volume 7, pages 1 – 15. Electronic Communication of the EASST, 2007.

[268] Harald Störrle. Semantics and Verification of Data Flow in UML 2.0 Activities. *Electronic Notes in Theoretical Computer Science, Elsevier*, 127(4):35 – 52, 2004.

Department of Computer and Information Science
Linköpings universitet

Dissertations

Linköping Studies in Science and Technology
Linköping Studies in Arts and Science

Linköping Studies in Statistics
Linköpings Studies in Information Science

Linköping Studies in Science and Technology

No 14 **Anders Haraldsson:** A Program Manipulation System Based on Partial Evaluation, 1977, ISBN 91-7372-144-1.

No 17 **Bengt Magnhagen:** Probability Based Verification of Time Margins in Digital Designs, 1977, ISBN 91-7372-157-3.

No 18 **Mats Cedwall**: Semantisk analys av process-beskrivningar i naturligt språk, 1977, ISBN 91- 7372-168-9.

No 22 **Jaak Urmi:** A Machine Independent LISP Compiler and its Implications for Ideal Hardware, 1978, ISBN 91-7372-188-3.

No 33 **Tore Risch:** Compilation of Multiple File Queries in a Meta-Database System 1978, ISBN 91- 7372-232-4.

No 51 **Erland Jungert:** Synthesizing Database Structures from a User Oriented Data Model, 1980, ISBN 91-7372-387-8.

No 54 **Sture Hägglund:** Contributions to the Development of Methods and Tools for Interactive Design of Applications Software, 1980, ISBN 91-7372-404-1.

No 55 **Pär Emanuelson:** Performance Enhancement in a Well-Structured Pattern Matcher through Partial Evaluation, 1980, ISBN 91-7372-403-3.

No 58 **Bengt Johnsson, Bertil Andersson:** The Human-Computer Interface in Commercial Systems, 1981, ISBN 91-7372-414-9.

No 69 **H. Jan Komorowski:** A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation, 1981, ISBN 91-7372-479-3.

No 71 **René Reboh:** Knowledge Engineering Techniques and Tools for Expert Systems, 1981, ISBN 91-7372-489-0.

No 77 **Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982, ISBN 91- 7372-527-7.

No 94 **Hans Lunell:** Code Generator Writing Systems, 1983, ISBN 91-7372-652-4.

No 97 **Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983, ISBN 91-7372-660-5.

No 109 **Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984, ISBN 91-7372-801-2.

No 111 **Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984, ISBN 91-7372- 805-5.

No 155 **Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons, 1987, ISBN 91-7870-133-3.

No 165 **James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning, 1987, ISBN 91-7870-183-X.

No 170 **Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems, 1987, ISBN 91-7870-225-9.

No 174 **Johan Fagerström:** A Paradigm and System for Design of Distributed Systems, 1988, ISBN 91-7870-301-8.

No 192 **Dimiter Driankov:** Towards a Many Valued Logic of Quantified Belief, 1988, ISBN 91-7870-374-3.

No 213 **Lin Padgham:** Non-Monotonic Inheritance for an Object Oriented Knowledge Base, 1989, ISBN 91-7870-485-5.

No 214 **Tony Larsson:** A Formal Hardware Description and Verification Method, 1989, ISBN 91-7870-517-7.

No 221 **Michael Reinfrank:** Fundamentals and Logical Foundations of Truth Maintenance, 1989, ISBN 91-7870-546-0.

No 239 **Jonas Löwgren:** Knowledge-Based Design Support and Discourse Management in User Interface Management Systems, 1991, ISBN 91-7870-720-X.

No 244 **Henrik Eriksson:** Meta-Tool Support for Knowledge Acquisition, 1991, ISBN 91-7870-746-3.

No 252 **Peter Eklund:** An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies, 1991, ISBN 91-7870-784-6.

No 258 **Patrick Doherty:** NML3 - A Non-Monotonic Formalism with Explicit Defaults, 1991, ISBN 91-7870-816-8.

No 260 **Nahid Shahmehri:** Generalized Algorithmic Debugging, 1991, ISBN 91-7870-828-1.

No 264 **Nils Dahlbäck:** Representation of Discourse-Cognitive and Computational Aspects, 1992, ISBN 91-7870-850-8.

No 265 **Ulf Nilsson:** Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs, 1992, ISBN 91-7870-858-3.

No 270 **Ralph Rönnquist:** Theory and Practice of Tense-bound Object References, 1992, ISBN 91-7870-873-7.

No 273 **Björn Fjellborg:** Pipeline Extraction for VLSI Data Path Synthesis, 1992, ISBN 91-7870-880-X.

No 276 **Staffan Bonnier:** A Formal Basis for Horn Clause Logic with External Polymorphic Functions, 1992, ISBN 91-7870-896-6.

No 277 **Kristian Sandahl:** Developing Knowledge Management Systems with an Active Expert Methodology, 1992, ISBN 91-7870-897-4.

No 281 **Christer Bäckström:** Computational Complexity of Reasoning about Plans, 1992, ISBN 91-7870-979-2.

No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.

No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.

No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2

No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.

No 338 **Simin Nadjm-Tehrani:** Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.

No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ ABB 1968-1993, 1995, ISBN 91-7871-494-X.

No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.

No 383 **Andreas Kågedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.

No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.

No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.

No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.

No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.

No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.

No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.

No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.

No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.

No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.

No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.

No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.

No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms, 1997, ISBN 91-7871-857-0.

No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.

No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.

No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.

No 485 **Göran Forslund:** Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.

No 494 **Martin Sköld:** Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.

No 495 **Hans Olsén:** Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.

No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.

No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.

No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Languages from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.

No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.

No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.

No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.

No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.

No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-x.

No 555 **Jonas Hallberg:** Timing Issues in High-Level Synthesis, 1998, ISBN 91-7219-369-7.

No 561 **Ling Lin:** Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.

No 563 **Eva L Ragnemalm:** Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.

No 567 **Jörgen Lindström:** Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.

No 582 **Vanja Josifovski:** Design, Implementation and Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.

No 589 **Rita Kovordányi:** Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.

No 592 **Mikael Ericsson:** Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.

No 593 **Lars Karlsson:** Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.

No 594 **C. G. Mikael Johansson:** Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.

No 595 **Jörgen Hansson:** Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.

No 596 **Niklas Hallberg:** Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.

No 597 **Vivian Vimarlund:** An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.

No 598 **Johan Jenvald:** Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.

No 607 **Magnus Merkel:** Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.

No 611 **Silvia Coradeschi:** Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.

No 613 **Man Lin:** Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.

No 618 **Jimmy Tjäder:** Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.

No 627 **Vadim Engelson:** Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.

No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.

No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.

No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.

No 688 **Marcus Bjäreland:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.

No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.

No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN-91-7373-126-9.

No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91 7373 207 9.

No 725 **Tim Heyer:** Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91 7373 208 7.

No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91 7373 212 5.

No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91 7373 258 3.

No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.

No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.

No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.

No 747 **Anneli Hagdahl:** Development of IT-supported Interorganisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.

No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-318-0.

No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.

No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.

No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.

No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002, ISBN 91-7373-349-0.

No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.

No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.

No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.

No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.

No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems´ Development, 2003, ISBN 91-7373-604-X

No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informationssystem, 2003, ISBN 91-7373-618-X.

No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.

No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.

No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.

No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.

No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing – An Empirical Study in Software Engineering, 2003, ISBN 91-7373-779-8.

No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.

No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.

No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.

No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.

No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.

No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.

No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.

No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5

No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004, ISBN 91-7373-966-9.

No 887 **Anders Lindström:** English and other Foreign Linguistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.

No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modelling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.

No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.

No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.

No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.

No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.

No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.

No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.

No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.

No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.

No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.

No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN: 91-85297-97-6.

No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.

No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.

No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005, ISBN 91-85457-07-8.

No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.

No 974 **Dan Lawesson:** An Approach to Diagnosability Analysis for Interacting Finite State Systems, 2005, ISBN 91-85457-39-6.

No 979 **Claudiu Duma:** Security and Trust Mechanisms for Groups in Distributed Services, 2005, ISBN 91-85457-54-X.

No 983 **Sorin Manolache:** Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour, 2005, ISBN 91-85457-60-4.

No 986 **Yuxiao Zhao:** Standards-Based Application Integration for Business-to-Business Communications, 2005, ISBN 91-85457-66-3.

No 1004 **Patrik Haslum:** Admissible Heuristics for Automated Planning, 2006, ISBN 91-85497-28-2.

No 1005 **Aleksandra Tešanovic:** Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components, 2006, ISBN 91-85497-29-0.

No 1008 **David Dinka:** Role, Identity and Work: Extending the design and development agenda, 2006, ISBN 91-85497-42-8.

No 1009 **Iakov Nakhimovski:** Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.

No 1013 **Wilhelm Dahllöf:** Exact Algorithms for Exact Satisfiability Problems, 2006, ISBN 91-85523-97-6.

No 1016 **Levon Saldamli:** PDEModelica - A High-Level Language for Modeling with Partial Differential Equations, 2006, ISBN 91-85523-84-4.

No 1017 **Daniel Karlsson:** Verification of Component-based Embedded System Designs, 2006, ISBN 91-85523-79-8

No 1018 **Ioan Chisalita:** Communication and Networking Techniques for Traffic Safety Systems, 2006, ISBN 91-85523-77-1.

No 1019 **Tarja Susi:** The Puzzle of Social Activity - The Significance of Tools in Cognition and Cooperation, 2006, ISBN 91-85523-71-2.

No 1021 **Andrzej Bednarski:** Integrated Optimal Code Generation for Digital Signal Processors, 2006, ISBN 91-85523-69-0.

No 1022 **Peter Aronsson:** Automatic Parallelization of Equation-Based Simulation Programs, 2006, ISBN 91-85523-68-2.

No 1030 **Robert Nilsson:** A Mutation-based Framework for Automated Testing of Timeliness, 2006, ISBN 91-85523-35-6.

No 1034 **Jon Edvardsson:** Techniques for Automatic Generation of Tests from Programs and Specifications, 2006, ISBN 91-85523-31-3.

No 1035 **Vaida Jakoniene:** Integration of Biological Data, 2006, ISBN 91-85523-28-3.

No 1045 **Genevieve Gorrell:** Generalized Hebbian Algorithms for Dimensionality Reduction in Natural Language Processing, 2006, ISBN 91-85643-88-2.

No 1051 **Yu-Hsing Huang:** Having a New Pair of Glasses - Applying Systemic Accident Models on Road Safety, 2006, ISBN 91-85643-64-5.

No 1054 **Åsa Hedenskog:** Perceive those things which cannot be seen - A Cognitive Systems Engineering perspective on requirements management, 2006, ISBN 91-85643-57-2.

No 1061 **Cécile Åberg:** An Evaluation Platform for Semantic Web Technology, 2007, ISBN 91-85643-31-9.

No 1073 **Mats Grindal:** Handling Combinatorial Explosion in Software Testing, 2007, ISBN 978-91-85715-74-9.

No 1075 **Almut Herzog:** Usable Security Policies for Runtime Environments, 2007, ISBN 978-91-85715-65-7.

No 1079 **Magnus Wahlström:** Algorithms, measures, and upper bounds for Satisfiability and related problems, 2007, ISBN 978-91-85715-55-8.

No 1083 **Jesper Andersson:** Dynamic Software Architectures, 2007, ISBN 978-91-85715-46-6.

No 1086 **Ulf Johansson:** Obtaining Accurate and Comprehensible Data Mining Models - An Evolutionary Approach, 2007, ISBN 978-91-85715-34-3.

No 1089 **Traian Pop:** Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies, 2007, ISBN 978-91-85715-27-5.

No 1091 **Gustav Nordh:** Complexity Dichotomies for CSP-related Problems, 2007, ISBN 978-91-85715-20-6.

No 1106 **Per Ola Kristensson:** Discrete and Continuous Shape Writing for Text Entry and Control, 2007, ISBN 978-91-85831-77-7.

No 1110 **He Tan:** Aligning Biomedical Ontologies, 2007, ISBN 978-91-85831-56-2.

No 1112 **Jessica Lindblom:** Minding the body - Interacting socially through embodied action, 2007, ISBN 978-91-85831-48-7.

No 1113 **Pontus Wärnestål:** Dialogue Behavior Management in Conversational Recommender Systems, 2007, ISBN 978-91-85831-47-0.

No 1120 **Thomas Gustafsson:** Management of Real-Time Data Consistency and Transient Overloads in Embedded Systems, 2007, ISBN 978-91-85831-33-3.

No 1127 **Alexandru Andrei:** Energy Efficient and Predictable Design of Real-time Embedded Systems, 2007, ISBN 978-91-85831-06-7.

No 1139 **Per Wikberg:** Eliciting Knowledge from Experts in Modeling of Complex Systems: Managing Variation and Interactions, 2007, ISBN 978-91-85895-66-3.

No 1143 **Mehdi Amirijoo:** QoS Control of Real-Time Data Services under Uncertain Workload, 2007, ISBN 978-91-85895-49-6.

No 1150 **Sanny Syberfeldt:** Optimistic Replication with Forward Conflict Resolution in Distributed Real-Time Databases, 2007, ISBN 978-91-85895-27-4.

No 1155 **Beatrice Alenljung:** Envisioning a Future Decision Support System for Requirements Engineering - A Holistic and Human-centred Perspective, 2008, ISBN 978-91-85895-11-3.

No 1156 **Artur Wilk:** Types for XML with Application to Xcerpt, 2008, ISBN 978-91-85895-08-3.

No 1183 **Adrian Pop:** Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages, 2008, ISBN 978-91-7393-895-2.

No 1185 **Jörgen Skågeby:** Gifting Technologies - Ethnographic Studies of End-users and Social Media Sharing, 2008, ISBN 978-91-7393-892-1.

No 1187 **Imad-Eldin Ali Abugessaisa:** Analytical tools and information-sharing methods supporting road safety organizations, 2008, ISBN 978-91-7393-887-7.

No 1204 **H. Joe Steinhauer:** A Representation Scheme for Description and Reconstruction of Object Configurations Based on Qualitative Relations, 2008, ISBN 978-91-7393-823-5.

No 1222 **Anders Larsson:** Test Optimization for Core-based System-on-Chip, 2008, ISBN 978-91-7393-768-9.

No 1238 **Andreas Borg:** Processes and Models for Capacity Requirements in Telecommunication Systems, 2009, ISBN 978-91-7393-700-9.

No 1240 **Fredrik Heintz:** DyKnow: A Stream-Based Knowledge Processing Middleware Framework, 2009, ISBN 978-91-7393-696-5.

No 1241 **Birgitta Lindström:** Testability of Dynamic Real-Time Systems, 2009, ISBN 978-91-7393-695-8.

No 1244 **Eva Blomqvist:** Semi-automatic Ontology Construction based on Patterns, 2009, ISBN 978-91-7393-683-5.

No 1249 **Rogier Woltjer:** Functional Modeling of Constraint Management in Aviation Safety and Command and Control, 2009, ISBN 978-91-7393-659-0.

No 1260 **Gianpaolo Conte:** Vision-Based Localization and Guidance for Unmanned Aerial Vehicles, 2009, ISBN 978-91-7393-603-3.

No 1262 **AnnMarie Ericsson:** Enabling Tool Support for Formal Analysis of ECA Rules, 2009, ISBN 978-91-7393-598-2.

No 1266 **Jiri Trnka:** Exploring Tactical Command and Control: A Role-Playing Simulation Approach, 2009, ISBN 978-91-7393-571-5.

No 1268 **Bahlol Rahimi:** Supporting Collaborative Work through ICT - How End-users Think of and Adopt Integrated Health Information Systems, 2009, ISBN 978-91-7393-550-0.

No 1274 **Fredrik Kuivinen:** Algorithms and Hardness Results for Some Valued CSPs, 2009, ISBN 978-91-7393-525-8.

No 1281 **Gunnar Mathiason:** Virtual Full Replication for Scalable Distributed Real-Time Databases, 2009, ISBN 978-91-7393-503-6.

No 1290 **Viacheslav Izosimov:** Scheduling and Optimization of Fault-Tolerant Distributed Embedded Systems, 2009, ISBN 978-91-7393-482-4.

No 1294 **Johan Thapper:** Aspects of a Constraint Optimisation Problem, 2010, ISBN 978-91-7393-464-0.

No 1306 **Susanna Nilsson:** Augmentation in the Wild: User Centered Development and Evaluation of Augmented Reality Applications, 2010, ISBN 978-91-7393-416-9.

No 1313 **Christer Thörn:** On the Quality of Feature Models, 2010, ISBN 978-91-7393-394-0.

No 1321 **Zhiyuan He:** Temperature Aware and Defect-Probability Driven Test Scheduling for System-on-Chip, 2010, ISBN 978-91-7393-378-0.

No 1333 **David Broman:** Meta-Languages and Semantics for Equation-Based Modeling and Simulation, 2010, ISBN 978-91-7393-335-3.

No 1337 **Alexander Siemers:** Contributions to Modelling and Visualisation of Multibody Systems Simulations with Detailed Contact Analysis, 2010, ISBN 978-91-7393-317-9.

No 1354 **Mikael Asplund:** Disconnected Discoveries: Availability Studies in Partitioned Networks, 2010, ISBN 978-91-7393-278-3.

No 1359 **Jana Rambusch**: Mind Games Extended: Understanding Gameplay as Situated Activity, 2010, ISBN 978-91-7393-252-3.

No 1373 **Sonia Sangari**: Head Movement Correlates to Focus Assignment in Swedish,2011,ISBN 978-91-7393-154-0.

No 1374 **Jan-Erik Källhammer**: Using False Alarms when Developing Automotive Active Safety Systems, 2011, ISBN 978-91-7393-153-3.

No 1375 **Mattias Eriksson**: Integrated Code Generation, 2011, ISBN 978-91-7393-147-2.

No 1381 **Ola Leifler**: Affordances and Constraints of Intelligent Decision Support for Military Command and Control – Three Case Studies of Support Systems, 2011, ISBN 978-91-7393-133-5.

No 1386 **Soheil Samii**: Quality-Driven Synthesis and Optimization of Embedded Control Systems, 2011, ISBN 978-91-7393-102-1.

No 1419 **Erik Kuiper**: Geographic Routing in Intermittently-connected Mobile Ad Hoc Networks: Algorithms and Performance Models, 2012, ISBN 978-91-7519-981-8.

No 1451 **Sara Stymne**: Text Harmonization Strategies for Phrase-Based Statistical Machine Translation, 2012, ISBN 978-91-7519-887-3.

No 1455 **Alberto Montebelli**: Modeling the Role of Energy Management in Embodied Cognition, 2012, ISBN 978-91-7519-882-8.

No 1465 **Mohammad Saifullah**: Biologically-Based Interactive Neural Network Models for Visual Attention and Object Recognition, 2012, ISBN 978-91-7519-838-5.

No 1490 **Tomas Bengtsson**: Testing and Logic Optimization Techniques for Systems on Chip, 2012, ISBN 978-91-7519-742-5.

No 1481 **David Byers**: Improving Software Security by Preventing Known Vulnerabilities, 2012, ISBN 978-91-7519-784-5.

No 1496 **Tommy Färnqvist**: Exploiting Structure in CSP-related Problems, 2013, ISBN 978-91-7519-711-1.

No 1503 **John Wilander**: Contributions to Specification, Implementation, and Execution of Secure Software, 2013, ISBN 978-91-7519-681-7.

No 1506 **Magnus Ingmarsson**: Creating and Enabling the Useful Service Discovery Experience, 2013, ISBN 978-91-7519-662-6.

No 1547 **Wladimir Schamai**: Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool, 2013, ISBN 978-91-7519-505-6.

No 1551 **Henrik Svensson**: Simulations, 2013, ISBN 978-91-7519-491-2.

No 1559 **Sergiu Rafiliu**: Stability of Adaptive Distributed Real-Time Systems with Dynamic Resource Management, 2013, ISBN 978-91-7519-471-4.

No 1581 **Usman Dastgeer**: Performance-aware Component Composition for GPU-based Systems, 2014, ISBN 978-91-7519-383-0.

No 1602 **Cai Li**: Reinforcement Learning of Locomotion based on Central Pattern Generators, 2014, ISBN 978-91-7519-313-7.

No 1652 **Roland Samlaus**: An Integrated Development Environment with Enhanced Domain-Specific

Interactive Model Validation, 2015, ISBN 978-91-7519-090-7.

No 1663 **Hannes Uppman**: On Some Combinatorial Optimization Problems: Algorithms and Complexity, 2015, ISBN 978-91-7519-072-3.

No 1664 **Martin Sjölund**: Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models, 2015, ISBN 978-91-7519-071-6.

No 1666 **Kristian Stavåker**: Contributions to Simulation of Modelica Models on Data-Parallel Multi-Core Architectures, 2015, ISBN 978-91-7519-068-6.

No 1680 **Adrian Lifa**: Hardware/ Software Codesign of Embedded Systems with Reconfigurable and Heterogeneous Platforms, 2015, ISBN 978-91-7519-040-2.

No 1685 **Bogdan Tanasa**: Timing Analysis of Distributed Embedded Systems with Stochastic Workload and Reliability Constraints, 2015, ISBN 978-91-7519-022-8.

No 1691 **Håkan Warnquist**: Troubleshooting Trucks – Automated Planning and Diagnosis, 2015, ISBN 978-91-7685-993-3.

No 1702 **Nima Aghaee**: Thermal Issues in Testing of Advanced Systems on Chip, 2015, ISBN 978-91-7685-949-0.

No 1715 **Maria Vasilevskaya**: Security in Embedded Systems: A Model-Based Approach with Risk Metrics, 2015, ISBN 978-91-7685-917-9.

Linköping Studies in Arts and Science

No 504 **Ing-Marie Jonsson:** Social and Emotional Characteristics of Speech-based In-Vehicle Information Systems: Impact on Attitude and Driving Behaviour, 2009, ISBN 978-91-7393-478-7.

No 586 **Fabian Segelström:** Stakeholder Engagement for Service Design: How service designers identify and communicate insights, 2013, ISBN 978-91-7519-554-4.

No 618 **Johan Blomkvist:** Representing Future Situations of Service: Prototyping in Service Design, 2014, ISBN 978-91-7519-343-4.

No 620 **Marcus Mast:** Human-Robot Interaction for Semi-Autonomous Assistive Robots, 2014, ISBN 978-91-7519-319-9.

Linköping Studies in Statistics

No 9 **Davood Shahsavani:** Computer Experiments Designed to Explore and Approximate Complex Deterministic Models, 2008, ISBN 978-91-7393-976-8.

No 10 **Karl Wahlin:** Roadmap for Trend Detection and Assessment of Data Quality, 2008, ISBN 978-91-7393-792-4.

No 11 **Oleg Sysoev:** Monotonic regression for large multivariate datasets, 2010, ISBN 978-91-7393-412-1.

No 13 **Agné Burauskaite-Harju:** Characterizing Temporal Change and Inter-Site Correlations in Daily and Subdaily Precipitation Extremes, 2011, ISBN 978-91-7393-110-6.

Linköping Studies in Information Science

No 1 **Karin Axelsson:** Metodisk systemstrukturering- att skapa samstämmighet mellan informationssystemarkitektur och verksamhet, 1998. ISBN-9172-19-296-8.

No 2 **Stefan Cronholm:** Metodverktyg och användbarhet - en studie av datorstödd metodbaserad systemutveckling, 1998, ISBN-9172-19-299-2.

No 3 **Anders Avdic:** Användare och utvecklare - om anveckling med kalkylprogram, 1999. ISBN-91-7219-606-8.

No 4 **Owen Eriksson:** Kommunikationskvalitet hos informationssystem och affärsprocesser, 2000, ISBN 91-7219-811-7.

No 5 **Mikael Lind:** Från system till process - kriterier för processbestämning vid verksamhetsanalys, 2001, ISBN 91-7373-067-X.

No 6 **Ulf Melin:** Koordination och informationssystem i företag och nätverk, 2002, ISBN 91-7373-278-8.

No 7 **Pär J. Ågerfalk:** Information Systems Actability - Understanding Information Technology as a Tool for Business Action and Communication, 2003, ISBN 91-7373-628-7.

No 8 **Ulf Seigerroth:** Att förstå och förändra systemutvecklingsverksamheter - en taxonomi för metautveckling, 2003, ISBN91-7373-736-4.

No 9 **Karin Hedström:** Spår av datoriseringens värden – Effekter av IT i äldreomsorg, 2004, ISBN 91-7373-963-4.

No 10 **Ewa Braf:** Knowledge Demanded for Action - Studies on Knowledge Mediation in Organisations, 2004, ISBN 91-85295-47-7.

No 11 **Fredrik Karlsson:** Method Configuration method and computerized tool support, 2005, ISBN 91-85297-48-8.

No 12 **Malin Nordström:** Styrbar systemförvaltning - Att organisera systemförvaltningsverksamhet med hjälp av effektiva förvaltningsobjekt, 2005, ISBN 91-85297-60-7.

No 13 **Stefan Holgersson:** Yrke: POLIS - Yrkeskunskap, motivation, IT-system och andra förutsättningar för polisarbete, 2005, ISBN 91-85299-43-X.

No 14 **Benneth Christiansson, Marie-Therese Christiansson:** Mötet mellan process och komponent - mot ett ramverk för en verksamhetsnära kravspecifikation vid anskaffning av komponentbaserade informationssystem, 2006, ISBN 91-85643-22-X.