Trusted Computing Engineering for Resource Constrained Embedded Systems Applications

Deliverable 7.1

Analysis of Challenges

Project:	TERESA
Project Number:	IST-248410
Deliverable:	D7.1
Title:	Analysis of Challenges
Version:	v1.5
Confidentiality:	Public
Editors:	Brahim HAMID
Date:	November 25, 2010

SEVENTH FRAMEWORK PROGRAMME

Part of the Seventh Framework Program
Funded by the EC - DG INFSO

Control Sheet

Approval		
	Name	Date
Prepared	(authors go here)	
Reviewed	All Project Partners	
Authorized	Antonio Kung	

Circulation	
Recipient	Date of submission
Project Partners	
European Commission	

Version History			
Version number	Date	Main author	Summary of changes
v1.0	01.04.2010	Brahim HAMID,Nicolas Desnos(IRIT)	Initial version
v1.2		Uni Siegen	added metrology section
v1.3	25.05.2010	Sigi Gürgens, Andreas Fuchs (SIT)	added pattern integration, introduction/conclusions parts
v1.4	14.06.2010	Brahim HAMID	Final version
v1.5	25.11.2010	Brahim HAMID	Confidentiality status correction

Contents

List of Figures

1 Introduction

The goal of this document is to advance the state-of-the-art in three issues that were identified within the context of TERESA as being of a long term interest from a scientific perspective as well as for Security and Dependability (S&D) engineering of embedded systems.

An embedded system is one that is composed of two main parts, software and hardware, and which are employed in a real world environment. Embedded systems concern numerous different domains such as aerospace, military, communications (e.g. mobile, GPS), medical, automotive, avionics, home control, industry control, and metering sector. Nowadays, with the progress of device technology, the embedded system domain is growing rapidly. However, the software of these systems cannot be built by using classical paradigms. Indeed, both software and hardware constraints must be taken into account. Furthermore, other problems like non-determinism and probabilities intervene.

The notion of Resource Constrained Embedded Systems (RCES) refers to systems which have memory and/or computational processing power constraints. They can be found in many application sectors such as automotive, aerospace, and home control. They are in many types of devices, like sensors, automotive electronic control units, intelligent switches, and home appliances such as washing machines and meters. In addition, they have different form factors, e.g. standalone systems, peripheral subsystems, and main computing systems. Computing resources of RCES, e.g. memory, tasks, and buffers, are generally statically determined. The development of RCES therefore involves specific software building processes. These processes are often error-prone because they are not fully automated, even if some level of automatic code generation or even Model-Driven Engineering (MDE) support is applied. Furthermore, many RCES also have assurance requirements, ranging from very strong levels involving certification (e.g. DO178 and IEC-61508 for safety-relevant embedded systems development) to lighter levels based on industry practices.

RCES are becoming increasingly complex and have various communication interfaces. Therefore, they have to be seen in the context of bigger systems or complete infrastructures. Consequently, their non-functional security and dependability requirements become more important as well as more difficult to achieve. The integration of S&D requires the availability of both application expertise and S&D expertise at the same time. In fact, S&D could also require both specific security expertise and specific dependability expertise. However, most organizations developing RCES have limited S&D expertise.

This document is organized as follows. The first part is dedicated to presenting the challenges and advances in enforcing S&D in RCES by model-driven engineering. The second part presents existing approaches for the integration and composition of systems (e.g.

cryptographic protocols). We discuss these approaches with respect to their usefulness for the integration and composition of S&D patterns in the context of security engineering for RCES. In the last part, we investigate the S&D related requirements of the metering sector which are candidates to be formalized. We also examine the conformity assessment procedures which could be applied to new metering devices developed using a trusted design process. Finally, we summarize the TERESA focus regarding WP7 and present a roadmap of the future work to be carried out in this work package.

2 Model Driven Trust Engineering for RCES

2.1 Context

2.1.1 Embedded Systems

An embedded system [1] is a system that is composed of two mainly parts, software and hardware, and which evolves in a real world environment. Embedded systems concern several domains like: aerospace, military, communication (mobile, gps,...), medical, automotive, avionic, home control, the industry control, the metering sector. Nowadays, the embedded system area is growing rapidly with device technology progress. Embedded systems are not classical software, which can be built by using classical paradigms. Indeed, software and hardware constraints must be taken in account together. Furthermore, other problems like non-determinism and probabilities intervene.

Each specific system has its own key property. For embedded system it is the real-time. Furthermore security, reliability, mobility, adaptive, heterogeneous systems and distribution must be also taken in account. In Teresa we focus on security and dependability. The implementation of security and dependability within embedded systems [2] is difficult because security attacks can be mounted at multiple levels of abstraction. In addition, the security cannot be applied as in classical software because these systems are resource-constrained in their capacities: consequently, it is easy for a hacker to access at the physical layer. Several competencies are required for building embedded systems: computer science (MDE, CBSE, combinatorial optimization, languages, compilers, concurrent systems), mechanical, electronics and security (cryptography, data access, etc.).

2.1.2 Resource constrained embedded systems (RCES)

Resource constrained embedded systems (RCES) refers to systems which have memory and/or computational processing power constraints. They can be found literally everywhere, in many application sectors such as automotive, aerospace, and home control. They are in many types of devices, like sensors, automotive electronic control units, intelligent switches, and home appliances such as washing machines and meters. In addition, they have different form factors, e.g. standalone systems, peripheral subsystems, and main computing systems. Computing resources of RCES, e.g. memory, tasks, and buffers, are generally statically determined. The generation of RCES therefore involves specific software building processes. These processes are often error-prone because they are not fully automated, even if some level of automatic code generation or even

model-driven engineering support is applied. Furthermore, many RCES also have assurance requirements, ranging from very strong levels involving certification (e.g. DO178 and IEC-61508 for safety-relevant embedded systems development) to lighter levels based on industry practices.

2.1.3 Security and Dependability in RCES

RCES are becoming increasingly complex and have various communication interfaces. Therefore, they have to be seen in the context of bigger systems or complete infrastructures. Consequently, their non functional requirements such as security and dependability (S&D) become more important as well as more difficult to achieve. The integration of S&D requires the availability of both application expertise and S&D expertise at the same time. In fact, S&D could also require both specific security expertise and specific dependability expertise. Most organizations developing RCES have limited S&D expertise.

2.1.4 Motivation

Modeling RCES are very important since their understanding and teaching are arduous. Several frameworks have been proposed to help the designers of such applications.

Embedded systems can be defined as information processing systems integrating hardware and software embedded into enclosing products to fulfill a specific function such mobile phone or fabrication equipment, lift control, defibrillator, etc. Co-design refers to the integration of both the software and the hardware parts at a high level to achieve functionalities, to describe architectural or behavioral aspects of a system [3, 4].

These systems come with a large number of common characteristics, including real-time and temperature constraints, dependability as well as efficiency requirements [5]. Specifically, resource constrained embedded systems (RCES) refers to systems which have memory and/or computational processing power constraints. They can be found in many application sectors such as automotive, aerospace, and home control. They are in many types of devices, like sensors, automotive electronic control units, intelligent switches, and home appliances such as washing machines and meters. In addition, they have different form factors, e.g. standalone systems, peripheral subsystems, and main computing systems. Computing resources of RCES, e.g. memory, tasks, and buffers, are generally statically determined. The generation of RCES therefore involves specific software building processes. These processes are often error-prone because they are not fully automated, even if some level of automatic code generation or even model driven engineering support is applied. Furthermore, many RCES also have assurance requirements, ranging from very strong levels involving certification (e.g. DO178 and IEC-61508 for safety-relevant embedded systems development) to lighter levels based on industry practices.

The development of these systems by traditional processes is no longer applicable and the need for new methods is necessary to design and provisioning software for resource constrained embedded systems correct by construction. The context of our works is to

build on two main currents in embedded systems research: software and hardware design of DRTES (Distributed Real Time Embedded Systems). We seek mechanisms that keep DRTES development safe, easier and quickly. In our works we aims at providing a framework combining models [6, 7, 8] and components [9, 10] to capture different facets of application requirements in terms of computation, memory, and those offered by the hardware. This is to lead a well negotiation scheme in the two wire constructions to achieve a system that meets both the requirements of the application and hardware resources. However, nor are such mechanisms fully unified in the sens that software and hardware design activities are done in a separate way. This separation may lead to software not adapted to a hardware target.

2.1.5 Summary

The rest of this part is organized as follows. In Section 2.2 we provide some reminders about modeling, tools, technologies and methodologies related to trust RCES applications. Section 2.3 recaps advances of the academic research in model driven development. In Section 2.4 we describe some standards devoted to build trust RCES applications Finally, section 2.5 concludes this. part with a classification of challenges related to TERESA.

2.2 Tools & Technologies

Models are used to denote some abstract representation of computing systems. Specifically, we need models to represent the network, models to encode protocols, software architecture and software platforms to test, to simulate and to validate the proposed solutions. In this section, we outline different tools used to describe several facets of S&D in RCES including process, patterns, component software engineering and model driven engineering.

2.2.1 Model-based development process

To cope with the growing complexity of embedded systems design, several developement approaches have been proposed. The most popular are those using models as the main artifacts to be constructed and maintained. In these processes, software development consists of model transformations.

The development process cycles are mainly iteratives, resulting in different levels of refining models of analysis and design. There are implementation platforms that adress these issues in specific contex (e.g, the MDA standard), but in many other contexts, the links between models refined or processed to solve references (to non-existent elements, elements not referenced, created elements,...) are still solved in ad hoc manner, without adequate support from generic technologies. The required solutions involve generally more reliable process, which essentially ensure consistency and traceability of produced models. We are still waiting for widely-applicable technologies that solve these problems in RCES environement.

2.2.2 S&D Patterns

A pattern deals with a specific, recurring problem in the design or implementation of a software system. It captures expertise in the form of reusable architecture design themes and styles, which can be reused even when algorithms, components implementations, or frameworks cannot. Today, design patterns are considered as fundamental technique to build software by capitalizing knowledge to solve occurring problems (in many specific domains). The design patterns for software building is derived from the Alexander's notion of patterns for Architecture [11] and a definition has been proposed by Buschmann in [12]: *A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution*. That is, patterns support the construction of software with defined functional and non functional properties. Design patterns are medium-scale patterns comparing to architectural patterns but they are at a higher level than the programming language. The application of a design pattern has no effect on the fundamental structure of a software system, but may have a strong influence on the architecture of a subsystem (components).

In this document we focus on the security aspects. Security is an important non functional requirement of software. In 1997, Yoder and Barcalow [13] was the first to work on security pattern documentation. A security pattern is a well- understood solution to a recurring information security problem. The typical structure of a security pattern is as follow [14]:

- Name: name of the security pattern
- Context: the security context describes the conditions where the security problem occur
- Problem: description of the problem
- Solution: techniques, structures and mechanisms to solve the problem
- Forces: define the types of trade-offs that must be considered in the presence of conflicts they might create
- Related patterns

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

In software engineering, patterns are considered as an efficient tool to reuse specific knowledge. For security & dependability we can encapsulate some experience in the design of such systems through the definition of specific design patterns. For instance, communication patterns are well suited to be used in embedded real time systems. Then, the implementation may be acheived using UML profiles.

The recurring appearance and use of some of these patterns led to build a catalog of patterns to encode the best practices of each filed. An interesting challenge is to address the problem of automation of the application of these design solutions. The difficulty is that the design solutions proposed by design patterns even differ in their details, while remaining similar in their principles. In addition, a design pattern is by construction not "complete", since it is devoted to allow many use according to small variations around the same filed.

Patterns for dependability. An hybrid set of pattern to be used in the development of fault-tolerant software applications is described in [15]. These patterns are based on classical fault tolerant strategies such as N-Version programming and recovery block, consensus, voting $\cdots$. In addition, the hybrid pattern structure can be constructed through recursive combination of N-Version programming and the others. This work addressed also the power of the technique through the support of the advanced software voting techniques. [16] proposed a framework for the development of dependable software systems based on a pattern approach. They reused proven fault tolerance techniques in form of Fault Tolerance Patterns. The pattern specification consists of a service-based architectural design and deployment restrictions in form of UML deployment diagrams for the different architectural services. The work is illustrated with an application to guide the self-repair of the system after the detection of a node crash.

Patterns for security. A collection of patterns to be used when dealing with application security is studied in [13]. The proposed catalog includes secure access layer, single access point, check point, etc.. The work of [17] reports an empirical experience, about the adopting and eliciting S&D patterns in the Air Traffic Management (ATM) domain, and show the power of using patterns as a guidance to structure the analysis of operational aspects when they are used used at the design stage. A survey of approaches to security patterns is proposed in [18].

The following figure depicts a set of most used patterns to secure Internet applications:

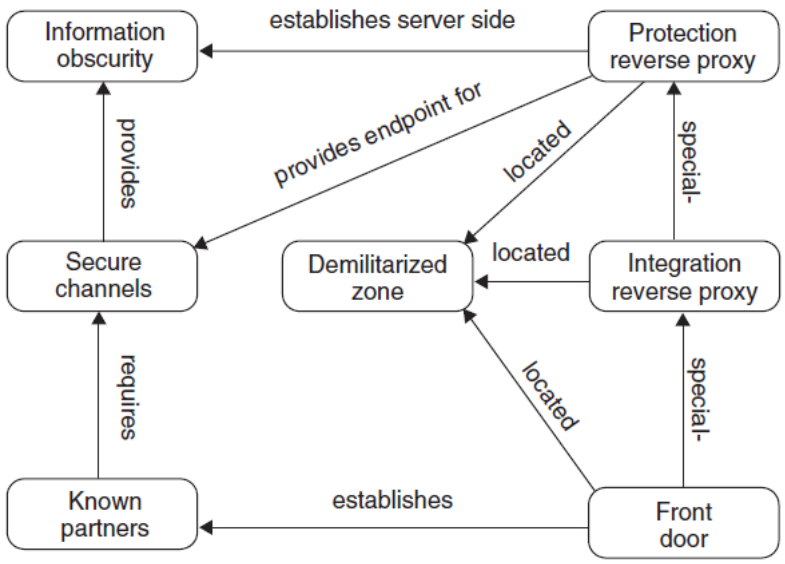

Figure 2.1: Example of some patterns to secure Internet applications

2.2.3 Component Software Engineering (CBSE)

Component-Based Software Engineering [19] allows build of large systems by assembling reusable components. It is a good solution to optimize the time and cost of software design while still guaranteeing the quality of the software. Moreover, the modularity it enables allows to tame the complexity of large systems. The life cycle of a component-based architecture can be divided into the following phases:

- Need analysis. Architect defines the set of requirements that the system will be able to meet.
- Design. An architecture is built to fulfill a set of requirements [20]. Usually, component-based approaches use formal languages called ADLs (Architecture Description Languages) to describe the structure and the behavior of a system. It allows working on high-level concepts without focus on the implementation. Medvidovic and Taylor have proposed a ADLs classification in [21].
- Validity. Most of systems verifies the correctness of the architecture; some also guarantee that all its requirements are met.
- Deployment and runtime. Deployment requires instantiating the architecture, configuring its physical execution context and dispatching the components in this physical context.
- Evolution. All software system must evolve [22].

Complexity in embedded system is rising due to richer functionality that is enabled by more powerful hardware. Component-based software engineering (CBSE) helps to capture several facets of this complexity, since (1) explicit functionalities may specified at a high level design to enable reuse, (2) the separation between a component specification and a component execution isolates the business logic from platform details and (3) a declarative deployment and configuration process facilitates the adaptation towards a specific target.

Usually, a component is seen as a black box that provides and requires services through its interfaces. Modeling component-based applications consists of describing components, their required and offered services and then define component instances and finally how these instances are connected to form the final system. At a specification level a system is described as a static interconnection of software component. At runtime a component assembly is an instantiation of an architecture composed of linked component instances. To help the design of trusted RCES applications, it seems usefull to use component as a first class artifact to implement software and hardware specifications and requirements.

2.2.4 Architecture Description Language

Another issue is ADLs instead of Architecture Description Languages. An ADL allows to describe the structure and the behavior of a component based system. In other words, it

describes how the components are assembled. Medvidovic and Taylor proposed an ADLs classification in [21]. Each ADL focuses on a specific objective. For example, Darwin [23] focuses on the component deployment description on distributed sites and Wright [24] focuses on the dynamic behavior of a system with formal methods. Moreover, the use of ADL allows the developer to concentrate only in the application level. For example, the developer presents only the threads, processes, and subprograms of his application. He doesn't interest of threads treating the communication of components. In addition, the ADLs have not only a power tool for presenting the software structure but also they allow the deployment and the configuration of components [25]. Adls can be classified following three kinds of schemes:

- Structural and behavioral scheme. an ADL allows to make a static description of the component classes as well as the static description of the connections between the components [21].
- Instantiating scheme. It describes the configuration of the system, at the component instances level. For example Wright [24] and Rapide [26] allow to describe a such schema.
- Deployment scheme. Darwin [23] has been built in order to facilitate the description of the building process of an application on many distributed sites. The semantic used by Darwin is based on the Pi-calculus and its distributed framework of execution is Regis.

2.2.5 Model Driven Engineering

The concept of model has becoming a major paradigm in software engineering. Its use represents a significant advance in terms of level of abstraction, continuity, generality, scalability, etc.. Model Driven Engineering (MDE) is a form of generative engineering [8], in which all or a part of an application is generated from models. It looks promising since it offers tools to deal with the development of complex systems improving their quality and reducing their development cycles. The development is based on model approaches, meta-modeling, development process and execution platforms.

As presented in [7], MDE may be considered form tow points of view: methodologists and developers. From methodologists an MDE process should define levels of abstraction, the modeling notations, the abstract syntax, how refinements are performed, how can a model be verified against the upper level model and how can it be validated. Developers consider the application of an MDE process as a models driven refinement steps.

The requirement for higher security, reliability and availability of systems is continuously increasing even in domains not traditionally strongly concerned by such issues. Required solutions are expected to be efficient, flexible, reusable on rapidly evolving hardware and of course at low cost.

Solutions are usually concrete technologies for a specific domain (avionics, transports and energy). Model-driven engineering provides a very useful contribution for the design of dependable and secure systems, since it bridges the gap between design issues

and implementation concerns (preoccupations). It helps the designer to concentrate on application structure and required behavior and permits to specify in a separate way non-functional requirements such as dependability/ fault tolerance and security as Quality of Service issues that are very important to guide the implementation process. That is, the model(s) can be analyzed at a very early stage in order to detect potential misconceptions; and then, exploited by specific tools through several steps of model transformation and/or interleaving with platform models in order to produce the application components and configuration files.

2.2.6 Integration Patterns in MDE processes

In a MDE approach, we need on one hand to capture non-functional aspect using some form of abstraction and in an another hand to be able to implement them. On way is to define patterns as a model types with abstraction execution support. In this way, patterns are applied to user models. The goal is to derive automatically design model from the specifications. The rogress from the specifications to an implementable model uses some patterns and the difficulties are how to choose the best pattern regardless of the specified constraints (efficiency, dependability, safety, footprint....) and the optimal and consistency composition of patterns.

On this basis, the activity of software design is aimed to move towards a detailed design model that will be directly implanted taking into account addition non-functional requirements (such as reliability, security, performance, scalability, maintainability, ..). In the object approach, the design is improved with the description of patterns (or motifs) of structures and interactions between objects. Such requirement are expressed separately for each object so the system resulting from the composition maintains the extra-functional properties required.

The design process can be seen as a progressive transformations of models into models corresponding to a successive applications of design-rules. Ideally, these rules should be expressed as design patterns. These rules are called "pattern language" in the sense that each designer has its role according to its knowledge (vocabulary) to solve a global problem.

The wanted role of pattern use is to ease, systematize and standardize the approach to the construction of software based systems. Most of the software architectures for large and complex systems have embedded patterns. However, the problem consists in identifying them explicitly.

The major purposes of using patterns is to develop software systems that are expected to provide the desired level of quality:

- Each pattern helps achieve one or more quality attribute in a system
- Each of them may also hinder other quality attributes

2.3 Current state of the academic research

The TERESA project addresses security issues for Model Driven Engineering (MDE). In this section, we describe projects that are closely related to TERESA.

2.3.1 Component Models

As defined in [9], a component represents an executable unit which can be deployed and composed at runtime. The component-based approach [27] allows to modulate the software part of embedded systems. It seems obvious that this approach allows the reuse of the software modules and the easy determination of the erroneous module in case of mistake. Moreover, the component-based approach facilitate the dynamism in embedded systems during its execution. Thanks to their modularity, it is easy to replace a component by another providing the same services.

Since many component models already exist – Lau et al. [28] compare for instance 13 of these. In the sequel, we investigate the most known component models that are explicitly suited for embedded systems.

Fractal. As depicted in Fig. 2.2, Fractal [29] is an extensible hierarchical component model that provides a powerful introspection mechanism based on the controller concept. It has been proposed by France Telecom R&D and INRIA in 2000 to tame the growing complexity of software systems. Its main objective is to build, to deploy and to maintain complex systems. A Fractal component is described as a black box that defines the services the component provides and requires through server and client interfaces and a content that allows a component to be recursively described. Fractal components are assembled into architectures by connecting client interfaces to server interfaces. Fractal proposes certain remarkable characteristics like a powerful introspection mechanism based on the controller concept and the sharing components.

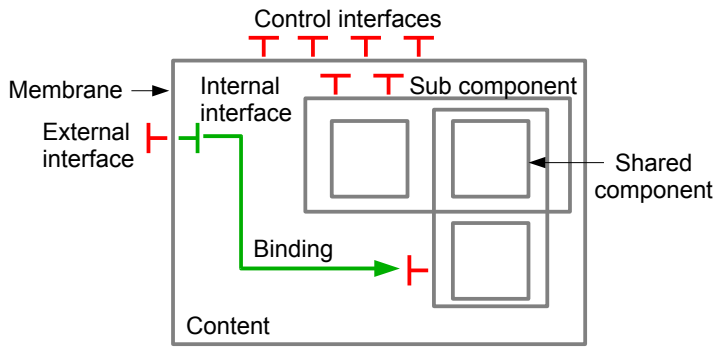

Figure 2.2: FRACTAL Component Model

SOFA. SOFtware Appliances is a hierarchical component model [30], based on behavior protocols, that focus on the architecture compliance checking [31] and that allows dynamic component substitution via its extension: Dcup (Dynamic Component Updating) [32]. A Sofa component has interfaces and uses behavior protocols, which are regular expressions that express the various possible sequences of events (traces) allowed by a component.

PECOS. PErvasive COmponent Systems component model was built in the context of an IST European project. Its goal is to build embedded systems by using a component technology while allowing to specify and check components. Component scheduling and memory consumption are the two main issues considered in this project. An important interest of this model is to check some non-functional requirements before the software be deployed in the hardware.

Koala. It is a component model dedicated to embedded devices. More precisely, Koala is used for building embedded software in TV sets (used at Philips). One of the advantages is to allow late binding of reusable components with no additional overhead.

ArchJava. It is a component model at the implementation level [33] where the separation between model and code does not exist. It represents a bridge between architectural specifications and the system itself, by enforcing communication integrity. An extension with the behavioral protocol concept is proposed in **Java**/**A** model [34].

2.3.2 Architecture Description Languages

Meta-H. It was developed by Honeywell, since 1991, under the sponsorship of DARPA and the US Army. It focuses on real-time avionics control software. It aims embedded systems in a bottom-up approach. In 2001, Meta-H has been taken like reference to build the AADL standard.

Wright. It was developed by Robert Allen and David Garlan, since 1997, at the Carnegie Mellon University. Wright [24] focuses on formal specification of software system architecture. The formalism is based on the CSP process algebra (Communicating Sequential Processes) to describe the dynamic behavior of ports of components. CSP has the advantage allowing to capture external and internal calls. Furthermore CSP has parallel composition and support theorem proving. It is also possible to detect dead-lock. Nevertheless CSP does not give possibilities to express timing constraints.

East-ADL. It was developed for dependable automotive embedded systems. The first version was proposed in the EAST-EAA project for the period 2002-2004 and a second version has been refined in the ATESST project for the period 2006-2008. East-ADL has been implemented as UML 2 profile.

Darwin. It was developed by Jeff Magge, since 1996, at the Imperial College London. Darwin [23] focuses on distributed systems. It allows to specify how component instances are distributed. The semantic used by Darwin is based on the Pi-calcul language and its execution platform is Regis. More precisely, Darwin allows to describe the collaboration between instances. It offers the possibility to specify the number of instances, or yet to add/remove instances.

Rapide. It was developed by David Luckham, since 1995, at the Stanford University. Rapide [26] focuses on formal aspects and events. Rapide takes in account the following concepts: components, connections and constraints. It defines and simulates behavior of distributed object architectures.

2.3.3 Modeling & development process

With regard to modeling and development process and platforms, we present in the following projects that are closely related to the expected engineering process of TERESA.

OpenEmbeDD. Model Driven Engineering open-source platform for Real-Time & Embedded systems (http://openembedd.inria.fr/) is an Eclipse-based "Model Driven Engineering" platform dedicated to Embedded and Real-Time systems (E/RT). It aims to offer engineers, who design and develop E/RT software the means to express, simulate, validate and test the targeted system before any component has soldered on a circuit board.

TopCased. The Open-Source Toolkit for Critical Systems project (http://www.topcased.org/) provides an open source framework for the development of critical embedded systems. It includes model editors, code generator, document generator, model based simulators, tools for formal verifications, ... Some of the tools (or part of them) have been developed in other research projects like OpenEmbeDD, SPICES, etc.

ModelWare. The ModelWare project aimed at defining and developing the complete infrastructure required for large-scale deployment of MDD (Model Driven Developpement) strategies and validating it in several business domains.

DOMINO. This project aims to define a methodology to build trusted components for model transformations. Components of that kind are used for modelling language translations (including Domain Specific Languages) and model to text transformation (document or code generation). Verification and validation aspects of such components are considered.

Usine logicielle -Software Company. This project is three-folded : modeling, validation and infrastructure/middleware support along with configuration support. The *Inflexion* subproject aims at providing a flexible infrastructure framework in the domain of real-time embedded applications. The framework promotes an infrastructure based on the CORBA Component model (CCM) and its Component/Container paradigm. It extends this model with the concept of Connector that provides an abstraction for connections. The project provides also tools to configure and generate efficient containers from a description of deployment specification at a model level. This work has already been successful in previous projects, namely the IST project Compare (http://www.ist-compare.org).

In this context, we proposed [35] a methodology associating model-driven approach and component based development to design distributed applications that has fault-tolerance requirements. UML based modeling is used to capture application structure and related non-functional requirements thanks to the complementary profile named FT profile which is composed of an extension of a subset of QoS&FT and uses NFP (Non Functional Properties) sub-profile of MARTE [36] (profile for Modeling and Analysis of Real-Time Embedded systems). Stereotypes dedicated to fault-tolerance specify the fault-detection policy, replication management style, replica group management. From this model we generate descriptor files (according to Deployment and Configuration standard (D&C)). These descriptor are in turn used to configure a devoted infrastructure consisting of a container/component based architecture, to build bootcode (static deployment) which instantiates, configures and connects components and to load configured components. Within this process, component replication and FT properties are declaratively specified at model level and are transparent for the component implementation.

SERINITY. The recently completed FP6 project SERENITY has introduced a new notion of S&D patterns. SERENITY's S&D patterns are precise specifications of validated security mechanisms, including a precise behavioral description, references to the S&D properties provided, constraints on the context required for deployment, information describing how to adapt and monitor the mechanism, and trust mechanisms. Such validated S&D patterns, along with the formal characterization of their behavior and semantics, can also be the basic building blocks for S&D engineering for embedded systems. [37] explains how this can be achieved by using a library of precisely described and formally verified security and dependability (S&D) solutions, i.e., S&D classes, S&D patterns, and S&D integration schemes.

ModelPlex. Modelling solution for complex software systems aims at defining and developing a coherent infrastructure specifically for the application of MDE to the devel-

opment and subsequent management of complex systems within a variety of industrial domains. ModelPlex uses established ModelWare technologies as a basis for providing an enhanced MDE ModelPlex approach. ModelPlex is driven by Industrial Use Cases ensuring the applicability and the integration of the different technologies produced by the academics and industrial partners.

OPEES. The mission statement of OPEES is "Ensuring long-term availability of innovative embedded software intensive systems engineering technologies to secure industry competitiveness and development'; its main goals are: (1) To build an ecosystem with the relevant business model that will ensure everlasting of embedded software industry aligned with the industries strategic intents; (2) To consolidate and share the results of R&D collaborative projects on System/Software engineering to ensure long term availability; (3) To consolidate & exploit requirements definition and experiences obtained by the industrial partners during R&D projects ; (4) To leverage early results and actors at the national level to give an European and worldwide dimension to the platform ; (5) To federate members proposals for new projects with the aim to ensure consistency and integration.

TimeAdapt. TimeAdapt [38] is a development process for the system design. It supports the execution of reconfigurations on component-based real time applications in a timely manner. The system design follows a three-tiered approach. TimeAdapt includes a reconfiguration specification language and a component model that supports dynamic reconfigurations on its elements. For each reconfiguration, a probability is estimated as to whether the given time bound can be meet. If the probability is high enough (as specified by the application), the reconfiguration is scheduled for execution as a real-time task.

Adapt.NET. Adapt.NET [39] is a framework for dynamic reconfiguration at middleware level using a descriptive and configuration language based in XML. Its implementation allows to identify the components and connections belonging to reconfiguration process and then to execute the appropriate reconfiguration commands. The execution of these commands in Adapt.NET requires the blocking of all connections between involved components. A transaction is blocked by waiting for all ongoing interactions to complete and also by not allowing the initiation of new transactions. In order to assure the consistency of system, the authors propose to reduce the blocking time by adding the components before the blocking time and deleting the components after. Moreover, The reconfiguration must be considered like an additional task that must be scheduled during the execution so that the tasks of application end in their deadlines.

SDCECM. SDCECM (Smart and Dynamic Customization Embedded Component based Model) [40] is a component model proposed in a framework for developing reconfigurable real time embedded systems. It can be dynamically configured and modified by allowing the developer to add or/and to modify a specific component or to fix a fault in its system.

The SDCECM components and the relations between them are described by XML. The structure of SDCECM component showed in Fig. 2.3 is composed by four levels:

- Component Manager Layer that manages internal and external behavior of a component. It includes the dynamic configuration, upgrade and replacement at runtime
- Component Descriptor Layer that describes the content of a given component
- Dynamic Component Implementation Interface (DCII) Layer that allows the dynamically reconfiguration of a component by allowing the customer to add, remove or replace an object and data during the execution of a system
- Static Component Implementation Interface (SCII) Layer that contains static or unconfigurable parts of component

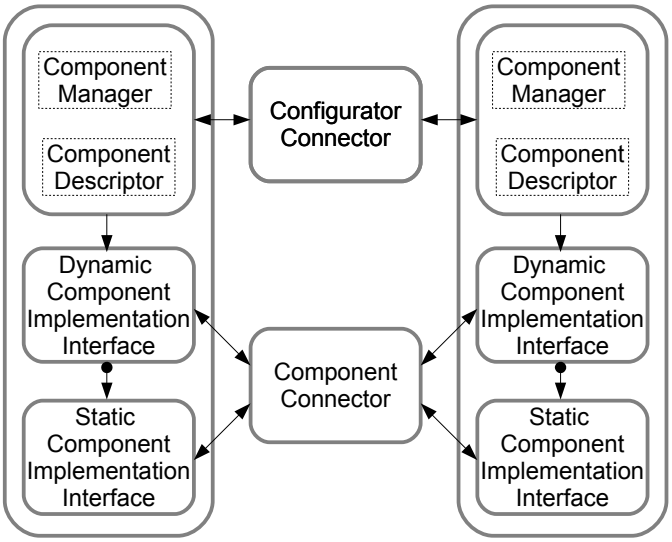

Figure 2.3: Layered structure of SDCECM

For the communication between the SDCECM components, two types of connector has been defined:

- Component connector that assures the communication between components
- Configurator connector that is an entity and presents a messaging tunnel between component managers when upgrading an existing component in a system

COMDES. COMDES (Component-Based Design of Software for Distributed Embedded Systems) [41] is a software framework for the specification and configuration of real time embedded systems. COMDES has been developed to address the design issues of distributed embedded systems, while taking account to the requirements of resource constrained applications. In this framework, the application is developed and configured from

prefabricated components, rather than generated. This framework defines two types of processes: configuration process and reconfiguration process. The configuration process allows to find components in the component repository and then to assemble them to configure an application model. A reconfiguration process allows adding, removing and updating components at runtime in order to update the application.

2.4 Standardization efforts

Here we investigate standards dealing with common modeling framework to design embedded systems including those form OMG [1] and SAE[2], that consider two types of knowledge: the partitions of the system and the level of abstraction.

2.4.1 Component Models

The components offer interfaces that allows to provide their non-functional properties and thus to predict the non-functional properties of global system [42, 10]. The non-functional properties are as important as the functional properties in the embedded systems. The component model allows the definition of structures, cycle of life, interaction methods and composition rules of components [25]. It can be characterized by a reflective and recursive behavior. In the sequel, we have a look at component models following the standardization effort that are explicitly suited for embedded systems: CCM and Autosar.

CORBA component model–OMG. A component definition consists principally of a definition of its ports and attributes. CORBA Component Model [43] (see 2.4, in short CCM supports five different kinds of ports: there is a pair of ports providing and requiring interface calls, called receptacle and facet. These ports exist in almost any component model and are called client and server in Fractal and required port (RPort) and provided port (RPort) in Autosar. In addition, CCM supports three ports for event based messaging: an emitter (single consumer), a publisher (n consumers) and a consumer. Events are delivered to the consumer in a push strategy. CCM offers a separation of business code located in the component from the nonfunctional or service code within a container. The CCM standard supports three different communication paradigms (port types): synchronous method calls based on CORBA (provided/required interface), event publishing and reception and the recently added streaming.

[1] Object Management Group
[2] Society of Automotive Engineers

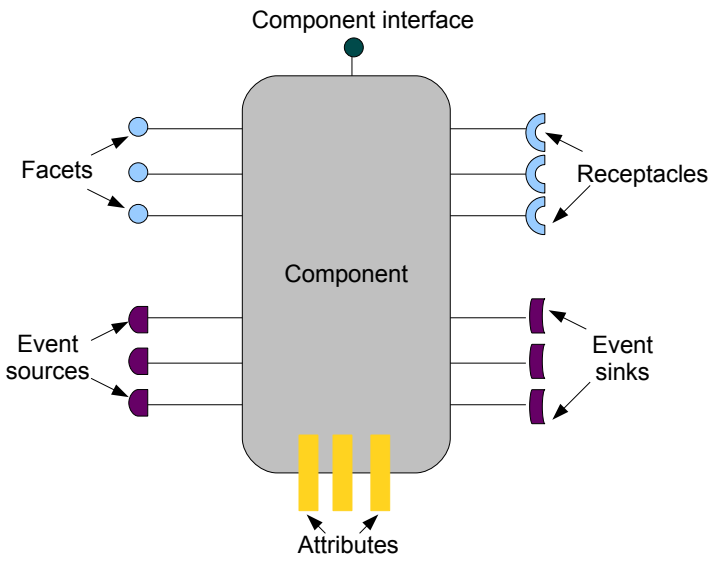

Figure 2.4: CORBA component model

Web services–OASIS. Distributed software components and independent that can be discovered and used via Internet usually named Web services appear more and more as a fundamental infrastructure for the electronic business process development like for example the electronic commerce. The standard formalism of a web service is described by its interface and is specified with the standard WSDL. The communication mode is described by the standard SOAP [44] (i.e. Simple Object Access Protocol). SOAP is a simple mechanism for transferring data that is based on the standard XML. It is possible to extend SOAP to add security. For instance, WS-Security is a communication protocol (SOAP extended) that allows to apply security to web services.

Autozar. Major automotive companies proposed Autozar [45] as a component model to describe and specify automotive applications. It has certain elements that are useful in the domain, e.g. a common set of infrastructure components (basic software), ports for dataflow and calibration. It is possible to model threads (runnables). Otherwise, the component model including deployment is not very different from CCM.

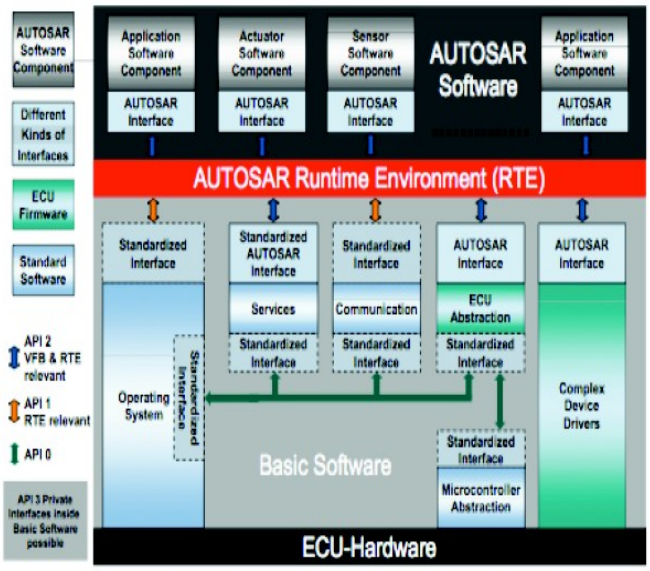

Figure 2.5: ECU Architecture

2.4.2 Language-based approaches

In the class of language-based approaches, relevant works include those of Architecture Description Languages and relevent SystemC works [46, 47, 48, 49].

AADL–SAE. Architecture Analysis and Design Language in short AADL [50] is based on the notion of component [21]. It defines an interface for each component and it allows the separation between the implementation of a component and the description of its interfaces. AADL describes both the software and the hardware parts of a system and allows the developer to concentrate only in the application level. For example, the developer presents only the threads, processes, and subprograms of his application. He doesn't interest of threads treating the communication of components.

AADL can be used for answering to the problematic according to the safety assessment of dependable systems and thus automating transformations of models. This one is motivated by the fact that the safety assessment of these systems is desirable to be explicitly

incorporated into the development cycle during the very early stages of a project. Combining AADL for designing the architecture and ontology-based model-driven engineering process for compositional safety analysis is a possible solution.

Fig. 2.6 presents an AADL specification which shows the component type called system and an implementation of this component. The interface of this component is specified by ports, required access to shared data, and flow of information through the component. The subsection "properties" shows the properties of this component. The implementation represents a composite component that has three subcomponents of system type.

Figure 2.6: Example of AADL GPS specifications

```
system GPS
features
   speed_data: in data port metric_speed
      {SEI::BaseType => UInt16;};
   geo_db: requires data access real_time_geoDB;
   s_control_data: out data port state_control;
flows
   speed_control: flow path
speed_data -> s_control_data;
properties SEI::redundancy => Dual;
end GPS;

system implementation GPS.secure
subcomponents
   decoder: system PGP_decoder.basic;
   encoder: system PGP_encoder.basic;
   receiver: system GPS_receiver.basic;
connections
   c1: data port speed_data -> decoder.in;
   c2: data port decoder.out -> receiver.in;
   c3: data port receiver.out -> encoder.in;
   c4: data port encoder.out -> s_control_data;
flows
speed_control: flow path speed_data -> c1 -> decoder.fs1
              -> c2 -> receiver.fs1 -> c3 -> decoder.fs1
```

```
              -> c4 -> s_control_data;
modes none;
properties arch::redundancy_scheme => Primary_Backup;
end GPS;
```

SystemC. A co-design environment based on the Java language which supports specification, co-synthesis and prototype execution for dynamically reconfigurable hardware/software systems is introduced in [46]. This is a design methodology to deal with adaptability and runtime-reconfigurability in networked embedded systems. Another issue is tooling around SystemC [48]. This is an ANSI standard C++ class library for hybrid systems design. It provides a mechanism for managing the complexity of complex systems that include large numbers of components by its facility for modeling hardware and software together at multiple levels of abstraction [47].

SystemC is a single, unified design and verification language that expresses architectural and other system-level attributes. It enables design and verification at the system level, independent of any detailed hardware and software implementation, as well as enabling co-verification with RTL design. This higher level of abstraction enables considerably faster, more productive architectural trade-off analysis, design, and redesign than is possible at the more detailed RT level. Furthermore, verification of system architecture and other system-level attributes is orders of magnitude faster than that at the pin-accurate, timing-accurate RT level [48]. Furthermore, SystemC is implemented as UML profile [49] to benefit from the power of UML modeling and to be used in MDA based approaches. The profile allows to build PIMs dedicated to SoC modeling, while the mappings defined are the base for transformation between PIMs and PSMs for code generation.

In the recent initiative [51] proposes a SystemC based executable modeling approach for the co design and early dependability assessment by means of simulated fault injection of safety critical embedded systems. Fault-tolerance is aimed at failure avoidance to deal with dependability in safety-critical embedded systems. The solution follows the international safety standard IEC-61508 for validating implemented fault-tolerance mechanisms. Thus, three consecutive abstraction levels are proposed for the design and simulation of fault injection: (1) At behavioral level, the designer implements the specified system behaviour and analyses how the system reacts in the presence of failures; (2) At architectural level, the designer analyses the dependability of the designed architecture; (3) At system implementation level, previous architectural elements are partitioned into hardware-software subsystems, where simulation based software and hardware fault injection is performed.

2.4.3 Modeling

Here we present UML, its several extensions devoted to deal with software architecture and non functional properties and tools dealing with the specification of constraints on UML models.

UML–OMG. The Unified Modeling Language (UML) [52] is the successor of the Object-Oriented Analysis and Design (OOA & D) methods that have emerged in the late 80s and early 90s. Since 1997, UML has become a standard of the OMG and adopted as a standard modeling language. UML 2.0 contains thirteen types of diagrams to model different views in order to represent specific concepts of systems:

- *Structural diagrams or static diagrams (UML Structure).* It contains class diagram, object diagram, component diagram, deployment diagram, package diagram and the composite structure diagram.
- *Behavioral diagrams or dynamic diagrams (UML Behavior).* It contains use case diagram, activity diagram, state diagram, transitions, interaction diagrams, sequence diagram, communication diagram, diagram of the overall interaction time chart.

These diagrams are not necessarily all used in a modeling. The most useful are activity diagrams, use cases, classes, objects, sequence and state-transitions. The component diagrams, deployment and communication are especially useful to formalize the implementation and the technical solution.

MARTE–OMG. This standard is inspired from the SPT profile [53] to support the specification, modeling and analyzing of real-time and embedded systems [36]. In fact, new requirements are considered: The separation in the specification of both the hardware and the software parts of platform resources, and the modeling of non-functional properties such as the time and the memory footprint. Such a profile adds constructs to describe the hardware and software (e.g., OS services) resources and defines specific properties to enable designers to perform timing and power consumption analysis.

MARTE [54] was proposed as a new UML profile to adds capabilities to UML for model-driven development of RTES. They also showed in [55] that MARTE UML profile is dedicated to characterize RTE multitasking execution platform via Software Resource Model (SRM) framework. It provides modeling artifacts to describe software execution platform modeling covering main multitasking capabilities such as real-time language libraries and real-time operating systems. This is to allow to specialize the generic resource concept to software domain, to describe concurrency support (e.g., tasks, interrupts and alarms), to detail interactions between concurrent resources (e.g., messaging, synchronization and mutual exclusion mechanisms) and to depict the software resource brokers (e.g., driver and scheduler). Furthermore, an open framework for detailed hardware modeling is presented in [56] as a dissemination of the Hardware Resource Model (HRM) part of the MARTE profile. This framework is composed of two views, a logical view that classifies

hardware resources depending on their functional properties, and a physical view that concentrates on their physical nature.

Extending these frameworks, [57, 53, 4] introduced the use of MARTE in a co-design methodology. In [57], they discussed a new co-design methodology to develop SoC/SoPC applications using three levels of abstraction: (1) Abstract Platform Architecture (extension of the GRM package of MARTE) (2) Execution Platform Architecture taking into account software and hardware nature of APA components and timing constraints and (3) Detailed Platform Architecture model to deal with other constraints as the communication protocol (extension of the HRM MARTE sub-profile). The first case study on modeling a RTES using MARTE is proposed in [53]. They evaluated the application of the profile to the Thales current systems (Josefil robotic system) through three stages: (1) a first logical decomposition, (2) SRM profile is applied in robot's sensors (POSIX RTOS) and (3) HRM profile is used for hardware architecture modeling. Recently, [4] describes a UML/MDA approach, called MoPCoM methodology, to design high quality real-time embedded systems. The MoPCoM co-design methodology is based on MARTE profile. This work details the Detailed Modeling Level which aims to perform automatically code generation from the model using three models: (1) The functional model allows to specify the behavior of the systems by means of an object oriented model, (2) The platform is a set of hardware components where behavior will resides and (3) The allocation maps the behavior onto the platform components. From the allocation model, the code generation tool extracts the new hardware components to be generated and writes VHDL code for each one.

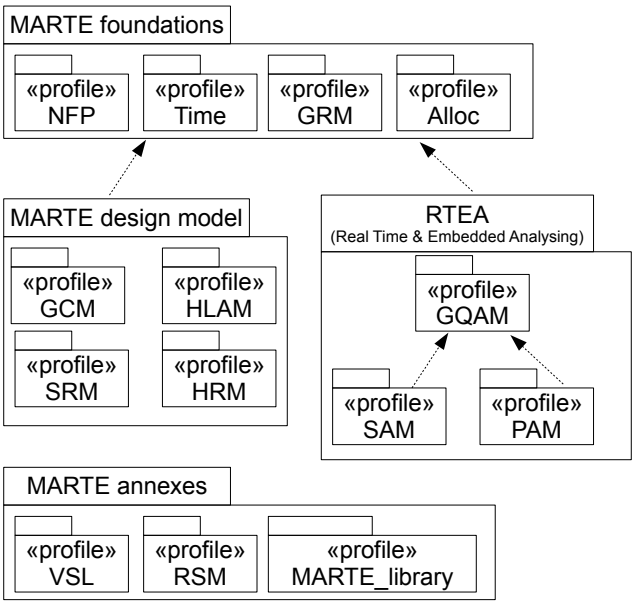

Figure 2.7: Structure of the MARTE Profile

SYSML–OMG. OMG poposed System Modeling Language in short SYSML [58] as a UML profile "for specifying, analyzing, designing, and verifying complex systems that may

include hardware, software, information, personnel, procedures, and facilities". The so-called Block concept is the common conceptual entity that factorizes many different kinds of system elements such as electronic or software components, mechanical parts, information units, and the description of different processes of systems. Blocks articulate a set of modeling perspectives enabling separation of concerns during systems design. However, SysML does not allow the strict Modeling of the temporal constraints and the resources management.

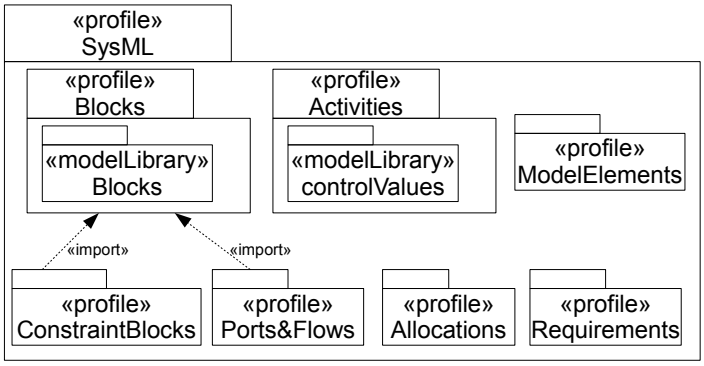

Figure 2.8: Structure of the SysML Profile

Object Constraint Language–OMG. This is an OMG standard for specifying all kinds of constraints. Concretely OCL is an expression language that extends UML for enriching it with constraints (specify class invariants, pre-and-post conditions of methods). In a certain point of view, OCL overcomes the lack of formalism proposed by UML: it bridges the gap between formal and semi-formal techniques. Indeed, UML gives a graphical model, like a class diagram. This is not sufficient to give a precise and unambiguous specification. OCL does not require a mathematical background. In other words, OCL has been built in order to be understood by the most of software developers, when other mathematical languages (based on mathematical concepts and notations) are too complex to be used. OCL is a modeling language, not a programming language. It allows expressing a set of constraints on a UML model, especially in class diagram and state-chart diagram. In the context of UML class-diagram, OCL is used to specify class invariants and to specify preconditions and postconditions of methods.

During the OCL expression evaluation the state of the objects in the system cannot change. In other words an OCL expression allows to make request on the state of the system without side effect.

Recently, QVT (OMG) introduced *ImperativeOCL* which extends OCL to manipulate system states through variables and to support programming language constructs such as loops and conditionals.

As shown in Fig. 2.9, OCL is composed of a set of packages:

- *OCL-AbstractSyntax.* Describes the syntactic structure. It is subdivided into two subpackages: the Expressions package defines the structure that OCL expressions can have. The Types package defines all the types used in OCL.
- *OCL-Domain.* Describes the values and evaluations. It is subdivided into two subpackages: The Values package describes the semantic domain. It shows the values OCL expressions may yield as result. The Evaluations package describes the evaluations of OCL expressions. It contains the rules that determine the result value for a given expression.
- *AS-Domain-Mapping.* Describes the associations of the values and evaluations with elements from the abstract syntax. It is subdivided into two subpackages: The Type-Value package contains the associations between the instances in the semantics domain and the types in the abstract syntax. The Expression-Evaluation package contains the associations between the evaluation classes and the expressions in the abstract syntax.

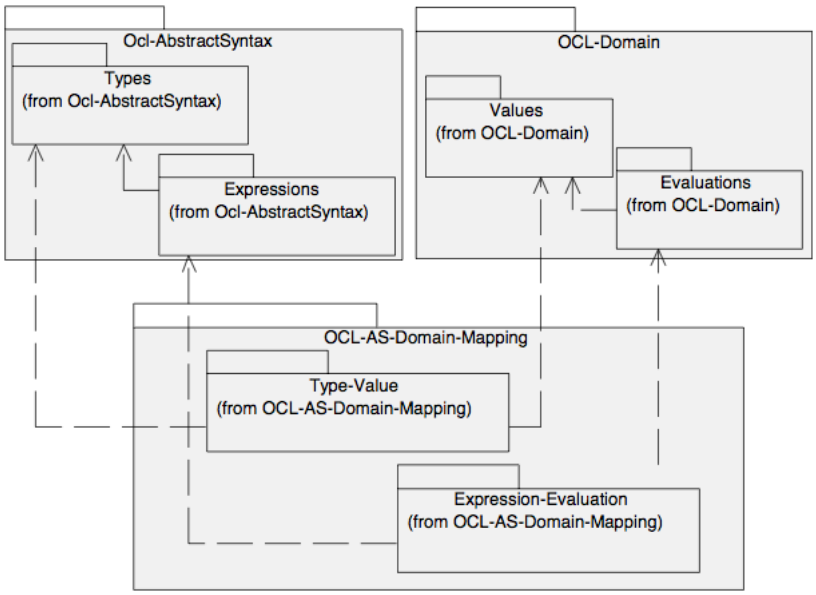

Figure 2.9: Structure of OCL packages

2.4.4 Developpement process

In this section we review briefly well known modeling software and system process.

SPEM–OMG. Modeling software and system process is fundamental in order to improve the quality of the application. In this way, SPEM (Software and Systems Process Engi-

neering meta-model) is a standard developed by the OMG for modelling of software process. The SPEM specification is used for describing a concrete software development process or a family of related software development processes. It uses the OMG's MOF meta-model and UML profiles.

The main goal of SPEM is to provide organizations with the means to define a conceptual framework. In a certain point of view SPEM can be seen as a sort of ontology of software development process. Thus, SPEM provides a formalize that allows structuring of processes. The necessary concepts for modeling, documenting, interchanging, and managing the development, are given by SPEM.

So, on the point of view of industrial actors SPEM constitutes an interesting way to modelling software and system process because it allows producing several documents like products, deliverables, guides, life-cycle, etc.

Concretely, SPEM includes a certain number of packages (see Fig. 2.10):

- *Core.* This package contains the meta-model classes and abstractions. It is the base for classes in all other meta-model packages.
- *Process structure.* This package defines the base for all process models. It represents a process as a static breakdown structure.
- *Process behaviour.* This package allows extending Process structure with behavioral models.
- *Managed Content.* This package allows to describe the content with models, and natural language descriptions.
- *Method Content.* This package provides the concepts for SPEM users. It gives possibilities to build up a development knowledge base that is independent of any specific processes and development projects.
- *Process With-Methods.* This package defines new and redefines existing structures for integrating processes defined with Process Structure meta-model package concepts with instances of Method Content meta-model package concepts.

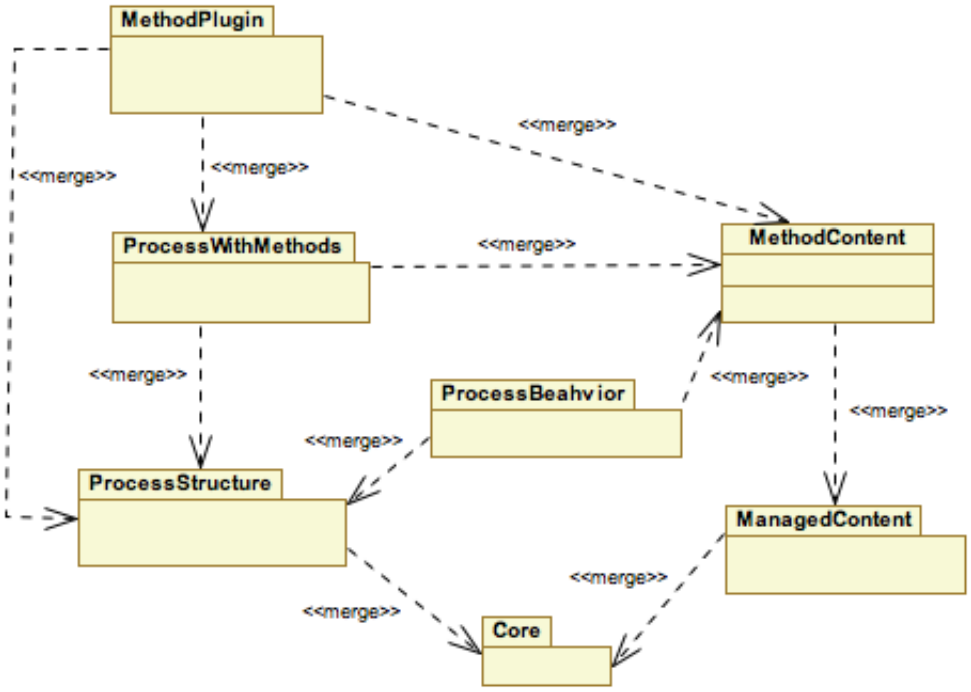

Figure 2.10: Structure of SPEM packages

MDA–OMG. OMG MDA (Model-Driven Architecture) [6] is a guide that is currently considered as a reference in the MDE community. It aimed at defining a framework of the MDE. Using UML formalism, the framework offers tools to create generic meta models and meta-models to a specific area or platform (UML profiles).

The MDA promotes a new way for the development of applications based on a Platform Independent Model (PIM) to transform into one or several Platform Specific Models (PSM), corresponding to each platform on which the application will be deployed. The MDA provides a development process summarized in the following steps:

1. Specify a system separately regardless of the platform that supports it, and thus to create a PIM,
2. Enrich the PIM model by successive stages,
3. Specify platforms,
4. Choose a platform for the system,
5. Transform the system specifications (PIM) in another specifications corresponding to a particular platform (PSM),
6. Refine the PSM to get an executable implementation (reification)

The three preliminary goals of MDA are: portability, interoperability and reusability through an architecture that promotes the separation of concerns. In the MDA process, all is considered as model. The four types of models used are: CIM, PIM, PSM and PDM.

- *CIM (Computation Independent Model).* it allows to specify the vision of a system in the environment where it operates, but without details of the structure of the system or its implementation. It helps to represent what the system will do exactly. The technical independence of this model allows it to keep its interest over time and is changed only if the knowledge or business needs change. The expertise is focused on the CIM specification rather than the technology implementation. In the construction of PIM (Platform Independent Model) and PSM (Platform Specific Model), it is possible to follow the requirements of CIM models that describe the situation where the system is used.
- *PIM (Platform Independent Model).* It is a model that is independent of any platform (EJB, CORBA,. NET,...) and doesn't contain information on technologies that will be used to deploy the application. It is a software model that represents a partial view of an CIM. It represents the functioning of instances and services. The frequency of its updating is relatively small and must be durable. It describes the system, but does not show the details of its use on the platform.
- *PSM (Platform Specific Model).* It is dependent on the technical platform specified by the architect. The PSM is essentially the basis for the generation of executable code to be load in this platform. There are several levels of PSM. The first is the result of the transformation of a PIM. It may be represented using UML diagrams related to a specific platform. Other PSM are obtained using successive transformations until the generation of a code in a specific language (C, Java, C#, etc.). In this case, a PSM contains information such as program code, related libraries and descriptors of deployment.
- *PDM (Platform Description Model).* It describes the platform on which the system will be implemented (for example models of components at different levels of abstraction: CCM, C#, EJB, EDOC, etc.). Currently, it is often specified in the form of software manuals and materials. In an MDA approach, PDMs are used to generate PSMs from PIMs.

The MDA identifies several changes during the development cycle. It is possible to make four different types of transformations:

.

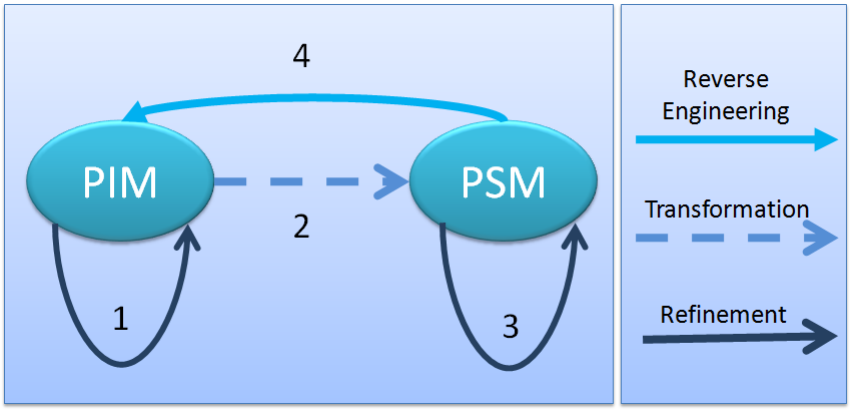

Figure 2.11: MDA Transformations

The transition from CIM to PIM is normally manual and subject to discussions:

1. *PIM to PIM.* this transformation allows to enrich, filter or specialize the model without using information about a platform. Such transformations are typically used for the refinement of the model.
2. *PIM to PSM.* this transformation starts when the PIM is sufficiently refined to be projected into a platform for execution. The characteristics of this platform may be described using a UML profile. The task consisting of adding information about a technical platform for code generation is a transformation PIM to PSM.
3. *PSM to PSM.* when a transformation PIM to PSM is not sufficient to code generation, we need additional transformation PSM to PSM using intermediary formalisms. For example, to generate C code from a UML formalism, a transformation UML to XML and XML to C could be used.
4. *PIM to PSM.* this transformation is used to return to a platform independent model (PIM) from a platform specific model (PSM) or possibly the code. It is a way to implement reverse engineering which is quite complex and difficult to achieve automatically.

Autosar Engineering Process. Fig. 2.12 depicts the process used in the automotive industry as recommended in the Autosar initiative (www.autosar.org) to generate the software elements of an electronic control unit (ECU). It includes a configuration phase and a generation phase. This process is common with other application sectors. They could be often simpler and sometimes involving a manual process.

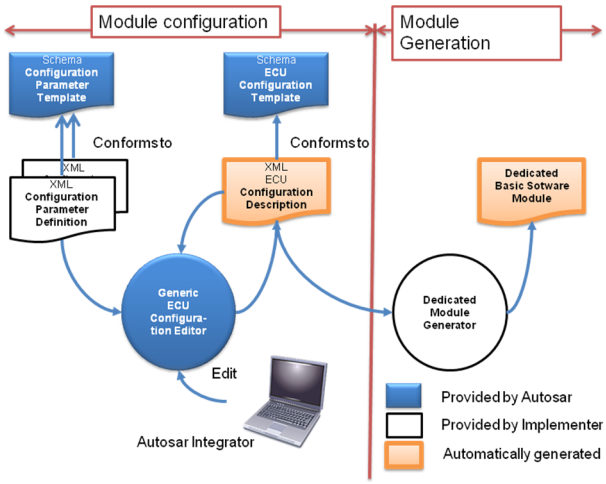

Figure 2.12: Structure of autozar process

V-Model–IEC 61508. The V-model [59] development process, also called verification & validation model, is suggested by the standard IEC 61508 (Standard for Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems). It is a trustworthy software development model, which aims at taming the complexity of project management, and which is used by big companies. The V-model appeared in the 1980s, then has been refined and used in the industry until today.

Concretely, the V-model focuses on verification and validation of the system at each step of the building process. Thanks to this model, a lot of potential errors that may be occur during the development are detect before. The V-model is inspired to the waterfall model. However, instead of moving down in a linear way, the V-model describes both the relationships and its associated step of testing between each step of the development life cycle. So, the main steps are the following:

- Requirement analysis: The requirements of the systems are identified by analyzing the needs of the users.
- System design: Designers analyze requirements and understand the business of the proposed system. Documentation about the software specification is made at this step.
- Software design: The structure of the system is elaborated. Several element are described like: the diagrams used, technology details, the classes, the modules, the functionalities, dependencies between modules, etc. Testing design is realized during this step.
- Coding: In this step the developers start coding. In order to make easy the coding, the designed system is broken up in to smaller units or modules.

The V-model has several advantages like the fact to detect errors in the early stage, or yet to give the opportunity to the developers to work in parallel in each side of *V* and also the fact that developers have a good understanding of the project during each step of the development. On the other hand, the V-model has several disadvantages like the fact that developers have not the possibility to build a prototype before the last step (because the software is developed in the implementation phase), or yet the development is less flexible and also costly in money and resource.

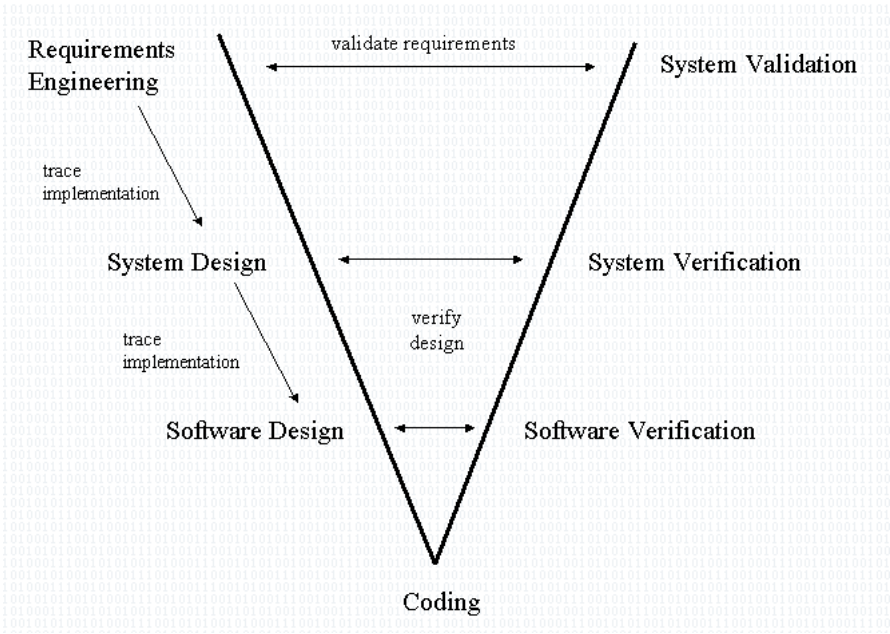

Figure 2.13: Structure of V-Model

Rup–IBM. The Rational Unified Process (RUP) is an iterative software development process framework, which aims at providing industry-tested practices for software and systems, and which has been created by the Rational Software Corporation (part of IBM since 2003). Concretely, the RUP process framework provides several fundamental elements like: processes tacking in account the best practices used in thousands of projects worldwide; capability patterns dedicated to the project managers for rapidly adding or removing reusable chunks of processes addressing common problems; and ready-to-use delivery processes dedicated to the project manager for planning and initiating a project.

The RUP process architecture is divided in four phases of development:

- Inception: The scope of the project and its business case are defined.
- Elaboration: The project's needs and the architectural foundation are defined.
- Construction: The application design and source code are created.
- Transition: The system is delivered to users.

Contrary to the V-model, RUP provides a prototype at the completion of each iteration. Within RUP process, each phase is organized into a number of separate iterations that must be satisfied before the next phase is undertaken.

One of the main advantage of RUP is to provide each team member documents (guidelines, templates and tool mentors) to take full advantage of among others the following best practices:

- Develop software iteratively: iterative approach; increasing understanding of the problem.
- Manage requirements: RUP shows how to organize, and to document required functionality and constraints.
- Use component-based architectures: Based on robust executable architecture.
- Visually model software: Shows how to capture the structure and the behavior of architectures and components.
- Verify software quality: RUP assists the developer in the planning, design, implementation, execution, and evaluation of these test types.
- Control changes to software: RUP shows how to control, track and monitor changes to enable successful iterative development.

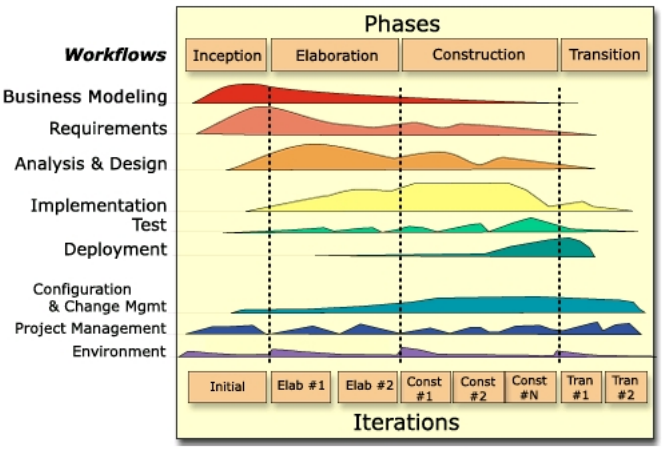

Figure 2.14: Architecture of RUP

2.5 Conclusions and Challenges

Model-Driven Engineering (MDE) provides a very useful contribution to the design of RCES applications since it bridges the gap between design issues and implementation concerns. Some of the most achieved works in this direction was those proposed in the context of the standards and research projects of Section 2.4 and Section 2.3. These standards allows the modeling of real time embedded systems by the specification of both the software and the hardware parts and the description of different processes of systems. Still a single modeling language is not suitable for modeling all aspects in embedded system (because of the varying nature of the disciplines involved in embedded system design). For example, SysML does not allow the strict modeling of the temporal constraints and the resources management.

In another hand, the goal of the academic research projects is to propose a common modeling framework to design embedded systems. Note, however, that these works doesn't exhibit sufficient maturity to apply these standards directly since they involve specific software building processes and no flexible levels of abstraction. In fact, their use in industry requires some expertise on MDE and specific domains. In addition, we require some instrumentations and support frameworks including encoding and more formal validation tools. Furthermore, by construction they are devoted for different purposes and follow different design rationales. In particular, managing security and dependability in embedded systems constitutes a difficult challenge because most of them interact with real world entities. Challenges for building secure embedded systems are numerous (real-time processing, security, reliability, power consumption, robustness) [60, 61, 62, 63]. Managing their ever-increasing complexity is one of the most important challenges [5]. Indeed, complex systems are constituted by many embedded systems (devices) that belong to several different domains and that use a mix of different technologies. The study of both hardware and software design for embedded systems is the key point for managing and understanding of such systems [63].

From the point of view of trusted RCES applications, the skills that will be required in the future for industrial systems will be in the fields of software design, execution platform, hardware integration and methodologies. The challenges dealing with S&D include real-time response, asynchronous communications, distribution, dynamic environment, performances, and recovering for failures [5] and so on.

2.5.1 Software design

Designing real time embedded systems is a very hard challenging task. The main difference with a classical software system is the fact that such systems must interact with real world entities. So a lot of disciplines must be taken in account in order to make the design in a good way. This is a systems-software co-design problem: functional requirements, non-functional requirements and physical constraints must be met. Several software design challenges are described as follow:

Need to use a common design language [62]. Using classical paradigms of computer science for building embedded systems is not suitable because many disciplines intervene in the building process: Software engineering, Mechanical and Electronics. Make a bad choice can be catastrophic (time and cost of development and material lost, or worse life lost). So, it is fundamental to make the good choice at high level of abstraction. The classical solution to design a specific embedded system is to use UML profile for each domain. Nevertheless, this classical solution is not usable in the "real world". Indeed, a single profile cannot capture all aspects in the multidisciplinary domain of embedded systems. Other solutions consist at combining profiles in a synergistic manner. The problem is that to combine two profiles can be inconsistent in certain contexts. Consequently, it lacks of common design language.

Robustness versus performance. Robustness and performance combine critical system engineering and best-effort engineering [5, 2]. Critical system engineering can be seen as a CSP (Constraint Satisfaction Problem) and a best-effort engineering can be seen as an optimization problem.

Dynamic system behavior. Many new embedded systems require dynamic behaviors. Indeed, most of embedded systems evolve in dynamic environments, and describing, then checking the correctness of their interactions is fundamental in order to obtain a correct embedded system. This one constitutes a difficult challenge (we find also this challenge in CBSE). Today there is not a unified theory to predict the dynamic properties of a software running on a given execution platform.

Distributed systems. Distributed systems research [64] covers a wide spectrum of applications including embedded real time systems, commercial transaction systems, transportation systems, avionic systems to name a few. A distributed system is a system which involves several computers, processors or processes which cooperate in some way to do some task. However, such systems raise several issues not found in single processor systems. The main difficulty arises since determinism is not required in the execution of distributed algorithms. In such systems, processors may have different speeds and links may have different delays, the different runs of the same algorithm on the same set of inputs will have different executions. So, it is a hard task to predict the execution of a distributed algorithm. Still, many paradigms in distributed computing system including: *election, naming, broadcast, mutual exclusion, synchronization and fault-tolerance*.

Security and Dependability in RCES. RCES are becoming increasingly complex and have various communication interfaces. Therefore, they have to be seen in the context of bigger systems or complete infrastructures. Consequently, their non functional requirements such as security and dependability (S&D) become more important as well as more difficult to achieve. For instance, to achieve transparent fault-tolerance, the system should be provide a service despite the occurrence of faults and under a certain strategy. It is a

very important property in the real world where failures can occur because of environment is changing.

Verification, validation, communication integrity. System integrity must be respected [65, 66]. This one means that the interactions that occur between the system components and the constraints imposed by the system platform and the application requirements must be met. It is an important challenge for embedded systems. So, efficient modeling frameworks have to provide a basis for analysis, optimization and validation. Communication integrity is an important challenge for embedded critical systems. The verification of embedded systems that support fault tolerance is very important since it makes possible to demonstrate that a system is correct even in the presence of faults and failures. On another side, validation effort increases exponentially with the number of components.

Adaptivity. As specified by T. Henzinger and J. Sifakis in [5], a great challenge at design time is to reduce the increasing gap between 2 diverging approaches and technologies: critical systems engineering and best-effort engineering. So this one consists to optimize using of resources while meeting both critical and best effort properties under uncertainty.

2.5.2 Execution platforms

At runtime must be taken in account both the environment and the needs of embedded systems to be autonomous. Indeed, many elements like temperature, wind, ??re, atmospheric pressure, or yet water, can impact the performance and the well working device. Furthermore, most of embedded systems evolve in dynamic environment. In this context they must be exible and adaptable like never before. Consequently, classical reconfiguring process at runtime is less suited to such systems. New strategies for self-management must be taken in account.

Autonomous systems. Autonomic computing [67] is a recent area that addresses the growing complexity of such systems by realizing tasks that are usually made by human. Autonomic computing has been introduced by IBM, since 2001, for taming this complexity. Such approaches are based on self-approaches and can be divided following four mainly parts [68]: self-Configuration (configures itself according to a set of objectives given at high level); self-Healing (detects and diagnoses problems), self-Optimization (optimizes its use of resources), and self-Protection (protects itself from malicious attacks - detects hostile behavior and provides an adapted answer).

Guaranteeing security & dependability while guaranteeing optimizing performance. Certain embedded systems are safety critical applications like nuclear, chemical plants, etc. In this case these systems require a high level of dependability while guaranteeing optimizing performance [5, 2].

Distributed systems. Distributed computing refers to the execution of distributed algorithms on distributed systems. A distributed algorithm executes as a collection of sequential processes, all executing their part of the algorithm without centralized control.
One of the main difficulty is to determinate the execution of such algorithm. Indeed, processors may have different speeds and links may have different delays, the different runs of the same algorithm on the same set of inputs will have different executions. So, it is a hard task to predict the execution of a distributed algorithm. Still, many paradigms in distributed computing system including: *election, naming, broadcast, mutual exclusion, synchronization and fault-tolerance*. In addition, the computing system is characterized by some constraints about its communication system [69, 70]. In such systems, a problem is often formalized following some properties (specifications) as fairness, liveness, termination, closure and convergence.

Reactivity: Guaranteeing real-time. Usually, embedded systems have to operate in a reactive way for answering at a query sent by a user or another device. The response must be sent within a predefined period of time [5]. Thus, the worst-case performance for executing an operation must be taken in account. For example, this one is very important for flight controller. Thus the challenge explained by Philip Koopman is the following: Worst case design analyses without undue pessimism in the face of hardware with statistical performance characteristics.

Secure and dependable Real-Time Systems. A particularly challenging task is efficiently integrating security and dependability solutions within the restricted available design space for RCES [63].

Deterministic response . The set of embedded systems can be divided in two sets: soft real-time systems and hard real-time systems [71, 61]. Given a query, hard real-time systems need to guarantee a deterministic response using appropriate scheduling schemes.

Dynamic environment and Flexibility . The ability to change the functionality of the system without implying heavy development cost is an important challenge [61]. Another important challenge is according to the nature of the environment: indeed, most of embedded systems evolve in environments that are highly dynamic and configurable.

2.5.3 Hardware and software

Works dealing with managing both hardware and software in embedded systems address the following challenges:

One of the most difficult tasks in the field of embedded systems is the hardware and software design interaction, which can be seen as the fact to merge and to satisfy a set of constraints. Works dealing with managing both hardware and software in embedded systems [1, 66] address the following challenges:

Autonomy. Lot of embedded systems must be autonomic [61]: managing power consumption, dissipation and adaptability are important challenges. Indeed, autonomy and adaptability provide powerful concepts to tame the added complexity in next generation systems.

Geometry. Main challenges are about the device geometry [61]. In other words the device mustn't follow rectangular and planar geometries.

CPU & Power management (Power consumption and dissipation) Real-time operating system power consumption constitutes an important component of the system software layer. So its management is a great challenge that form a field of research RTOS [72]. Predicting the amount of resources required by embedded software is fundamental in order to verify that the system will meet its real-time and resource constraints. An interesting challenge proposed by Cabillic is to provide a resource consumption evaluation of embedded software (CPU, energy, or memory) for facilitating the development of embedded systems.

Adapted to the environment. Many embedded systems do not evolve in a controlled environment [61]. Furthermore, the environment can impact important damages to the device because of: atmospheric pressure, water, fire, vibration, lightning, power supply fluctuations, and corrosion.

Optimizing. Understanding links between hardware and software is important for applying dynamic scheduling policies respecting optimality criteria [61, 66].

2.5.4 Methodologies

In the context of embedded systems, another kind of great challenge is to define methodologies for describing specific kinds of embedded systems. A set of strategies is listed as follow:

Define a system engineering process for security and dependability for RCES. The integration of S&D requires the availability of both application expertise and S&D expertise at the same time. In fact, S&D could also require both specific security expertise and specific dependability expertise. Most organizations developing RCES have limited S&D expertise.

Generic vs. specific engineering. Building an infrastructure for large-scale deployment of model driven engineering strategies to specific domains.

Enforcing S&D in RCES with Model Driven Engineering. Model-Driven Engineering (MDE) provides a very useful contribution to the design of RCES applications since it bridges the gap between design issues and implementation concerns. Used properly, MDE can potentially maintain the separation of concerns between application and S&D, by ensuring that S&D designs can be reused at a later stage by application designers. Significant research is being carried out concerning MDE for embedded systems, at the level of system architecture, design techniques, testing, validation, proof of correctness, modeling, software reliability, operating systems, parallel processing and real-time processing. More research is needed on the use of MDE to enforce the integration of S&D requirements into the engineering process and to support the reuse of S&D mechanisms. One important focus in this field is on the potential benefits of the combination of model-driven engineering with pattern-based representation of security and dependability solutions. Of particular interest is related to the development of models and tools to support the inclusion of S&D issues into the RCES engineering process.

Define a methodology to build trusted components for model transformations The generation of RCES therefore involves specific software building processes. These processes are often error-prone because they are not fully automated, even if some level of automatic code generation or even model driven engineering support is applied. Furthermore, many RCES also have assurance requirements, ranging from very strong levels involving certification (e.g. DO178 and IEC-61508 for safety-relevant embedded systems development) to lighter levels based on industry practices.

3 S&D Pattern Integration

3.1 Introduction

The goal of the TERESA project is to develop a security engineering process that is adapted to resource constrained systems. This process will make extensive use of S&D patterns. These contain the specification of an S&D solution, the security property it provides, the assumptions on the environment that need to be satisfied in order for the security property to hold, and other important information that is needed in order to integrate the solution into an application. While work package WP5 is concerned with the development of methods that allow to verify that a particular solution indeed provides a particular S&D property, and to determine the assumptions under which this property holds, in Task 2 of work package WP7 we are concerned with the even more challenging question of solution integration and composition.

While a lot of research has been focussing on verifying a specific S&D solution in a closed world, i.e. without interference by other solutions running in parallel or consecutively, in real world security engineering, the closed world assumption is not valid. Solutions do run concurrently, and there is even the need to compose solutions, as complex S&D properties can often not be realised by one single security mechanism. In general, it can not be assumed that the resulting composition provides the desired S&D property. If for example an RSA encryption is combined with an RSA signature with message recovery (i.e. without using a hash function) and both mechanisms use the same key pair, the confidentiality property that shall be achieved by encryption is violated. This is a very simple example, for more complex combinations of solutions it is not as easy to recognize possible security issues. Furthermore, composition of solutions can result in a complex S&D property which is stronger than the union of the properties of the separate solutions if the solutions are composed “correctly”. The development of methods that result in “correct” composition and integration of solutions is the topic of Task 2 of workpackage WP7.

In this deliverable we will summarize the main research results that have been achieved with respect to integrating and composing S&D solutions.

3.2 State of the Art

Research on composition of systems mostly focusses on either design or verification. The design related approaches provide methods that allow to compose components providing certain properties to a system while preserving these properties. The approaches

focussing on verification aim at reducing the complexity of verifying that a particular system provides a particular property. This is achieved by decomposing the system into subsystems and verifying that these subsystems provide the desired properties, and by proving that the decomposition satisfies certain conditions that are proven to preserve the properties.

3.2.1 Approaches addressing Information Flow Properties

A huge amount of work has already been performed, addressing different types of properties being preserved under composition. One prominent area of research is concerned with information flow properties. Roughly, information flow properties describe that no information leakes from high-level events (performed by honest users) to low-level events (performed by an adversary), resulting in confidentiality properties, and that no information leakes from low-level events to high-level events, which corresponds to an adversary not being able to influence the actions of honest users, hence represents an integrity property.

In [73], Mantel introduces a uniform basis for compositionality results with respect to information flow properties, thus providing insight into relations between compositionality results that previously seemed unconnected. His approach can be used both for design and verification. Mantel specifies systems in terms of sequences of possible events and distinguishes between input events (controlled by the environment) and output and internal events (controlled by the system). A system is composed of components by synchronization of shared events. There are three different categories of events: visible events, events that are neither visible nor confidential, and confidential events. Mantel then defines Basic Security Predicates (BSP) in terms of traces of events. These BSP essentially capture that there are always sufficiently many traces so that an adversary cannot deduce information of a particular kind (depending on the respective property). One BSP for example states that if there is a trace including a confidential event, then there is another trace with the same visible events but without the confidential event which means essentially that what an intruder can deduce does not depend on whether or not the confidential event occurs. Mantel then can show that these BSP are equivalent to particular information flow properties addressed in other works (such as noninference, forward correctability, nondeducibility on outputs, etc.).

The *Generalized Zipping Lemma* then formulates sufficient conditions for specific Basic Security Predicates to be preserved under composition. These conditions require non-visible events of one system not to be events of the other system, and they further relate those subsets of visible, non-visible, and confidential events, respectively, that are addressed by particular BSPs to intersections of the visible, non-visible and confidential events of the two systems. Hence the zipping lemma describes the nature of the traces of the two systems regarding the occurence and sequence of visible, non-visible and confidential events that must / must not be shared by both systems.

As already stated above, Mantel's approach addresses information flow properties that can be used to specify confidentiality or integrity properties of a system. It is not clear

how other security properties (like freshness, non-repudiation) or trust assumptions that are needed for PKIs or Trusted Computing based technology can be addressed using this approach.

3.2.2 Universal Composability

This is another prominent branch of research addressing the composition of cryptographic protocols while preserving certain security properties (hence it falls into the "composition for design" category). In this context security of protocols is understood as guaranteeing, in the presence of an adversary, a set of correctness properties of the output values of the parties together with a set of secrecy requirements regarding the local data of the parties. The security of a protocol is proven by evaluating a probabilistic function of the parties' inputs (*secure function evaluation*). A common paradigm in this area of research is that a protocol that "securely realizes" its task is equivalent to running an idealized computational process (also called "ideal functionality") where security is guaranteed.

In [74] (see also [75] for a more comprehensive elaboration), Canetti introduces his *Modular Composition* approach. Here he describes the ideal process as follows: All parties hand their input to some incorruptable "trusted party" which computes the desired outputs and returns them to the parties. Hence the adversary in this setting is limited, it only learns the input and output of corrupted parties. Protocol parties are modeled as Interactive Touring Machines (ITM), the adversary controls a subset of the protocol ITMs (at most less than half of the parties can be corrupted at any time).

A protocol securely performs the task at hand if executing the protocol (in a given model of distributed computation) amounts to emulating the ideal process for that task. More specifically, the output of running a protocol with a given adversary in a given model is formalized, as well as the output of running the ideal process with an "ideal process adversary". Then, running the protocol emulates the ideal process if, for any adversary attacking the protocol in the given distributed model, there exists an "ideal process adversary" such that for any set of inputs to the parties, the global output of running the protocol with its adversary is indistinguishable from the global output of running the ideal process with its "ideal process adversary". This is proven by proving that the global outputs of the protocol and the ideal process, respectively, are equally distributed. This approach can address various different types of adversaries (active vs. passive ones, non-adaptive vs. adaptive ones, etc.).

Different communication models (e.g. for different systems/applications) are modeled by letting the parties access an appropriate ideal functionality that captures the guarantees (assumptions) provided by the relevant model. That is, the ideal functionality can be viewed as representing a physical assumption or an abstract property that is later realized by some other protocol. When the protocol shall for example ensure authenticity, i.e. that some entity P sends data d authentically to entity Q, the ideal process can be specified as follows: P hands d to the trusted party which forwards (P, Q, d) to the ideal-process adversary. Once the adversary returns an OK message, the trusted party forwards (P, d) to Q and halts. This way it is guaranteed that message d to Q is authentically sent by P. If

additionally confidentiality of the message is required, the trusted party does not disclose d to the adversary unless either the sender or the receiver is corrupted.

This definition is sufficient for capturing security for stand-alone protocols. In order to address protocols that are running with two or more instances and to model information flow between a protocol instance and the rest of the network, Canetti extends the Universal Composibility framework by adding an "environment machine": It chooses arbitrary input for all protocol instances and the adversary, and it collects their output, and it then announces whether it "thinks" to have interacted with the protocol or with the ideal process. The environment is allowed to interact with the adversary in any way. Now a protocol securely evaluates a function if for any adversary there exists an ideal adversary such that no environment can tell with non-negligible probability whether it is interacting with the protocol and its adversary or with the ideal process and the ideal adversary.

Canetti's *modular composition* methodology for secure protocols is based on the following idea:

- Design a "high-level" protocol for the given task assuming that other, simpler, subtasks can be carried out securely.
- Design protocols that securely carry out these simpler subtasks.
- Construct a full-fledged protocol for the given task by plugging the simpler protocols as subroutines in the "high-level" protocol.

More precisely, let p be a protocol that UC-realizes some ideal functionality $\mathcal{F}$. In addition, let π be some arbitrary protocol (regarded "high level protocol") whose parties both interact in the usual way and additionally make ideal calls to multiple instances of $\mathcal{F}$. To construct the composed protocol π^p, one starts with π and replaces each call to a new instance of $\mathcal{F}$ with an invocation of a fresh instance of p, a message sent to an existing instance of $\mathcal{F}$ is replaced by an input value given to the corresponding instance of p, output of an instance of p is treated as output of the corresponding instance of $\mathcal{F}$. The **Universal Composition Theorem** then states that running π^p, with no access to $\mathcal{F}$ is equivalent to running protocol π. That is, for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{A}_{\mathcal{F}}$ such that no environment machine can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running π^p or with $\mathcal{A}_{\mathcal{F}}$ and parties running π. In particular, if π securely realizes some ideal functionality $\mathcal{G}$ then π^p securely realizes $\mathcal{G}$. "Sequential" and "parallel" compositions are special cases of "modular composition".

Another approach that falls in the category of universal composability and focusses particularly on authentication and key exchange protocols is presented by Bellare et al. in [76]. This approach gives an idea of how protocols that UC-realize some ideal functionality can be designed. Its main idea is to design a solution that works in the "authenticated model", an idealized setting in which communication is per se authentic, and transform this solution into a realistic model with a Dolev-Yao type adversary by way of a so-called "MT-authenticators" (message transmission authenticators), and finally prove that it emulates the protocol in an unauthenticated network. While the solution in the authenticated model assumes e.g. authenticated channels, the MT-authenticator can be thought of as some primitive that makes use of well-known mechanisms (e.g. challenge response) and

appropriate cryptographic algorithms (e.g. a signature algorithm). For the proof that the MT-authenticator emulates the protocol in the authenticated model, an adversary is constructed that uses the protocol in the authenticated network and does everything that the adversary in the nonauthenticated network does with the MT-authenticator, so that all input and output of the protocol in the authenticated network and of the MT-authenticator have negligible statistical distance. This finally is shown by proving that an event that contradicts the authenticity property can only happen with negligible probability by showing that it is equivalent to breaking the signature scheme (or any other cryptographic algorithm that is used within the MT-authenticator).

A main disadvantage of the Universal Composability approach seems to be that for every property that a protocol shall provide, a new ideal process has to be constructed whose interactions with the parties result in providing this property. It is not clear whether this can be achieved for sufficiently many properties being required in particular for embedded systems. Further it is not clear whether this type of proofs relying on probabilistic assessment can be achieved for all these properties.

3.2.3 Logic-Based Approaches

A completely different type of approach is the Protocol Composition Logic (PCL) presented amongst others by Datta et al. in [77]. They introduce a modal logic with additional two operators identical to temporal logic that specify that some formula holds in some state in the past and that some formula holds in the previous state, respectively. Axioms and inference rules are formulated to capture security mechanisms. The main idea for proving that a protocol provides specific properties is as a first step to define a protocol template as a cord space (a form of process calculus), and to specify the security properties achieved by the protocol and the assumptions under which these properties hold. The concrete functions to be used by the protocol (like symmetric encryption, keyed hash, etc.) are substituted by function variables. In a second step the protocol template is instantiated by different concrete protocols, in particular the function variables are instantiated by concrete cryptographic algorithms, and it is proven that for these protocols, the assumptions hold. Hence the instantiations provide the same security properties as the template.

A refinement operation then adds an additional security property which results in the combination of two protocols. This is again achieved in two steps: First two protocol templates are defined and it is proven that each of them provides a specific property. Then substitutions have to be found for each of the function variables in these protocols such that the substitutions yield the same protocol, and finally it has to be proven that the instantiation satisfies the union of the assumptions for the two protocol templates.

Similarly to any other axiomatic approach, the axioms and inference rules that are used for the proofs capture the properties of the mechanisms being used in the protocols. Hence the reliability of such a proof depends on the accuracy of the axioms and inference rules. Further, for each new mechanism that shall be addressed, new axioms and inference rules need to be introduced.

In [78], Cremers identifies a number of problems with PCL. According to Cremers, PCL cannot be used to prove common authentication properties of protocols that do not include signatures. Further, he shows that a number of claimed proofs in PCL cannot be correct because there is no way to establish preceding actions in a thread, and there is no way to express type restrictions in PCL. According to Cremers, some of these problems can be resolved by minor modifications to PCL, but other problems require further investigation.

3.2.4 Strand Spaces

Another area of research is based on so-called Strand Spaces. A strand here is a sequence of message (so-called term) transmissions and receptions (so-called nodes) by a certain agent. A bundle is then a collection of strands that describe a certain system execution. It is represented as an ordered graph with two kinds of edges, one to describe interactions between different strands and a second to describe precedence within one strand. The attacker model for this kind of protocol validation is the so-called penetrator strand that has a certain set of capabilities to intercept and insert messages between the protocol participants as well as to perform certain concatenation and crypto operations – similar to Dolev-Yao.

Based on this notion Fábrega, Herzog and Gutman [79] describe the concept of mixed strand spaces. It allows for a security analyst who attempts to investigate a composed protocol to first analyse the simpler sub-protocols before judging on the mixing of them. The step of mixing protocols is that each node is attached to a primary or secondary strand, where each of these represent the two protocols to be mixed. In order to then prove that a protocol keeps e.g. its authentication property during a mixed run, it is sufficient to use the proof from the isolated investigation of the protocol and further to prove that terms originate from a primary node rather than just from a regular node in the unmixed run. This approach is closely related to the horizontal composability to be explained in Section 3.2.8. However the underlying formal basis in this case is the model-based notion of strand spaces rather than the notion of a transition system.

3.2.5 Protocol Independence

In [80], Andova et al. present an approach that can be used to compose protocols sequentially while preserving their security properties. The model for security protocols they use was introduced in [81]. Here they present a formal semantics based on sets of roles, agents (trusted and untrusted ones), variables, functions, and constants in which the functions are used to describe abstract properties of cryptographic primitives (e.g. $k^{-1}((m)) = m$). The formal semantics further contains a definition of what are events (in this paper the only events are send, read, and claim events, while for protocol composition they add create and end events). A protocol specification is constituted by a number of role specifications that consist in the initial knowledge of the particular role and its events.

Predicates are used to describe requirements on such a specification in order to be well-defined (e.g. such as a variable can not be sent without having been received), and to specify the acceptance of messages by agents (assuming typed messages).

The initial knowledge of the intruder contains the initial knowledge of all untrusted agents plus some data only the intruder knows. Different intruders can be modeled with different capabilities by way of rules (e.g. an intruder that can only forward messages which actually models a network without any intruder, that can only eavesdrop on messages, that can inject its own messages, etc.).

The formal semantic further defines the instantiation of a protocol, called a run, and how the protocol specification is transformed into a run specification. A state of a protocol is then defined as the product of the knowledge of an intruder, the input buffer (containing all receive events that are not yet processed), the output buffer (containing all send events not yet processed) and the remaining events that still have to be executed. A set of derivation rules specifies the possible state transitions of a protocol instantiation specified by this formal semantics.

Security properties are defined in terms of traces of the protocol specification, a trace being a sequence of states reachable by applying the derivation rules. A security claim is not global but local to one agent's run, i.e. it is a requirement held by one agent that a particular property holds). Secrecy for example is defined by essentially requiring that if a trusted agent requires a particular term to be secret in a particular state then in all preceding states, the intruder must not know this term. They further define *synchronisation*, a particular instantiation of authentication.

For proving a security property to hold for a protocol specification, they introduce some lemmas that are concerned with statements about terms as part of a state in a trace implying other (related) terms being or not being part of some other state of this trace and/or the intruder knowledge. These lemmas together with the derivation rules describing state transitions are then examplarily used to prove that the secrecy property with respect to the responder's nonce holds, and that the synchronisation property holds.

This approach is the basis of the work presented in [80] that introduces a very strong, but efficiently verifiable notion of independence between protocols:

Two protocols are independent if no encryption term produced by the first protocol running in the context of the second protocol will be decrypted or verified by the second protocol, and vice versa. In general, proving independence is a non-trivial problem, but for many protocol sets it is easy in the sense that the protocol sets satisfy an even stronger notion of independence: Two protocol sets are strongly independent if they have no encryptions of the same form. Unlike independence, strong independence can be easily verified at the syntactical level, and it implies independence. Further, different protocols can use the same cryptographic keys and still be strongly independent, and thus independent. Strong independence can be achieved by adding unique protocol tags to the protocol messages.

Using this notion of protocol independence, they prove that every protocol remains correct (i.e. its security properties are not violated) when running together with independent pro-

tocols. They address secrecy and protocol-centric properties (these are properties that depend on only one trace, authenticity being a typical member of this class of properties). Finally they prove that this type of security properties is preserved under sequential composition of strongly independent protocols.

The requirement of independency restricts this approach to a small set of realistic protocols. Further, while sequential composition of protocols is certainly of interest, more often the security properties of different protocols running in parallel has to be addressed.

3.2.6 Disjoint Encryption

In [82], Guttman introduces another approach that is based on a syntactic relation between protocols. He models protocols using strand spaces and protocol "skeletons" that he defines to be the parts of a protocol carried out by the honest agents. He defines two protocols Π_1 and Π_2 to satisfy *strong disjoint encryption* if Π_2 does not create encryptions of forms specified in Π_1 and further, if, when Π_2 execution receives a value only inside encryptions specified in Π_1, it does not re-transmit the value outside these encryptions. He then proves that for any two protocols Π_1 and Π_2 satisfying this condition, a protocol goal (which can be either confidentiality or authenticity properties) of Π_1 is preserved in $\Pi_1 \cup \Pi_2$.

This approach can very well be used for protocol design, as according to Guttmann strong disjoint encryption can easily be achieved by inserting a randomly chosen tag for each protocol and insert it in all (encrypted) messages. However, for the composition and integration of already existing protocols, strong disjoint encryption will in many cases be an assumption that does not hold. Further, the number of security properties that can be handled is quite limited (only authenticity and confidentiality).

3.2.7 Protocols as Compositions of Primitives

A further approach that at first sight looks promising was introduced by Choi in his PhD thesis [83]. He composes so-called protocol primitives in order to generate more complex security protocols. These primitives are essentially simple challenge-response protocols that include bindings, i.e. a nonce plus information to identify the initiator and responder of a protocol, respectively. Confidentiality and authenticity of the primitives are achieved by way of symmetric or asymmetric encryption, and hash functions. The primitives are shown to satisfy certain properties: the agreement property as defined by Lowe [84] (that essentially states that the two protocol participants agree on all relevant data exchanged in the protocol and on their respective communication partners), regularity (no exposition of long-term secrets), discreetness which essentially means that traces shall not expose any data that can be derived from a set of secrets and combinations of its elements, and nonces not being used for more than one participant.

Using strand spaces to model protocols he shows that primitives satisfying the above properties can be combined without violating their authenticity and confidentiality properties.

By extending the primitives by e.g. unique session identifiers he provides unambiguity of message parts with respect to belonging to a specific run of a protocol.

Ths approach is only useful in case completely new protocols shall be designed. It can not be used when composing already existing protocols. Further, it is limited to addressing only authenticity and confidentiality.

3.2.8 Transition System Based Channel Notions

Another branch of research utilizes transition systems as they have been known for quite some time and formulates properties based on these models that are better comprehensible than a formal expression. One possibility to do so is to introduce a notion of channels as e.g. Rudolph [85]. Whilst his work utilizes a notion from automaton theory and primarily focuses on the validation of single protocols, Mödersheim and Viganò [86] introduce a more detached notion for channels with security properties and further focus on the challenge of protocol composition.

The notion introduced by Mödersheim and Viganò is called AnB• notation and consists primarily of the following symbols [86]:

- $Authentic\ channel : A{\bullet\!\!\rightarrow}B : M$ represents an authentic channel from A to B. This means that B can rely on A having sent the message M.
- $Confidential\ channel : A{\rightarrow\!\!\bullet}B : M$. This means that A can rely that only B can receive the message M.
- $Secure\ channel : A{\bullet\!\!\rightarrow\!\!\bullet}B : M$. This is a channel that is both authentic and confidential.
- $Fresh\text{-}authentic\ channel : A{\bullet\!\!\twoheadrightarrow}B : M$. This channel is like an authentic channel with the restriction, intuitively, that M can only be received once.
- $Fresh\text{-}secure\ channel : A{\bullet\!\!\twoheadrightarrow\!\!\bullet}B : M$. A channel that is both fresh-authentic and confidential.

This rather abstract notion may be used in order to define protocols' *goals* as well as *assumptions*. Further there exists work from Mödersheim [87] to translate this notion to a more low-level description for the state-space transition system *AVISPA Intermediate Format IF* for more detailed analysis.

However, the AnB• notation can be used directly to describe protocols such as Diffie-Hellman similar to deductive rules:

$$\frac{\begin{array}{lcl} A & \bullet\!\!\rightarrow & B : exp(g,X) \\ B & \bullet\!\!\rightarrow & A : exp(g,Y) \\ A & \rightarrow & B : \{Data\}_{exp(exp(g,Y),X)} \end{array}}{A \quad \bullet\!\!\rightarrow\!\!\bullet \quad B : Data} \tag{P2}$$

This notion also gives rise to a rather intuitive way of visualizing composition of protocols. As example [86] provides a second protocol using digital signatures for authentic transmission of a message:

$$\begin{array}{cccc} A' & \rightarrow & B' & : \{B', M'\}_{inv(pk(A'))} \\ \hline A' & \bullet\!\!\rightarrow & B' & : M' \end{array} \tag{P1}$$

These two protocols can then be composed by matching of the goal of P1 with one of the assumptions of P2. In this case, they replace the first protocol step of P2 with P1 by mapping $[A' \mapsto A, B' \mapsto B, M' \mapsto exp(g, X)]$:

$$\begin{array}{cccc} A & \rightarrow & B & : \{B, exp(g, X)\}_{inv(pk(A))} \\ B & \bullet\!\!\rightarrow & A & : exp(g, Y) \\ A & \rightarrow & B & : \{Data\}_{exp(exp(g,Y),X)} \\ \hline A & \bullet\!\!\rightarrow\!\!\bullet & B & : Data \end{array} \tag{P2[P1]}$$

Aside from the representation of protocol composition, Mödersheim and Viganò also inspect the conditions under which such a composition may or may not lead to additional vulnerabilities. To describe this, they introduce the notion of *horizontal composability* that states that an attack against the parallel execution of a set of protocols implies an attack on the parallel execution of one protocol out of this set. Investigating this definition, it can be seen that if a set of protocols is horizontally composable, i.e. each of the protocols can be executed in parallel without introducing attacks, then the parallel execution of the whole set of protocols does not introduce attacks.

This definition is specialized to *vertical composability* that covers the case of composition of protocols, where the outer protocol and a variantion of the inner protocol have to be horizontally composable. This means that a message of the composed protocol cannot be mistaken for a message of each of the individual protocols.

This approach to verify the composability of protocols demonstrates the strong reliance of Mödersheim and Viganò that all protocol variables of the composed protocols are disjoint. As they state in the subsequent text, “we need to ensure that the messages and submessages of the protocols cannot be confused and that the behavior of P_1^* is independent from the concrete payload message, e.g. by using tagging.”

This rather strong assumption provides a very challenging problem for formal analysis of protocols in general. The basis for the analysis is built up from automaton theory and formal languages and therefore based on “strongly typed” input alphabets. Further the means of abstraction when formally specifying a real system (in order to reduce verification complexity) includes the reduction of several information to single representatives, which further introduces the necessity for strong typing.

For these reasons there exists the need for further scientific research at this point, as [86] among others demonstrates the dependence of verification approaches on such a property that yet has to be formally formulated and investigated rather than left for the formalization “expert” to decide manually.

3.2.9 A Generic Process for Composition

So far most approaches known and introduced here focus rather on the pure validation of compositions in those cases where the property of the resulting integration is known a priori. However the composition of protocols is a process that also includes the non-trivial step to derive the property that the composition of two separate solutions shall provide. This task becomes especially meaningful if the composed solution is supposed to provide a stronger property than each of the utilized solutions separately. Whilst often, the composition of protocols is used such that one protocol fulfills the requirements of another protocol – resulting in the second protocol's properties to be the composed property – there are cases where two protocols are utilized closer integrated and the protocols' properties are to be composed as well.

One attempt to formalize this process was presented by Fuchs, Gürgens and Rudolph [88]. The work describes a "Generic Process for Pattern Integration", where the term *integration* is used as a synonym for composition. It originates from the SERENITY project [89] and utilizes the Security Modelling Framework SeMF for the description of security properties and as formal basis for any kind of proof to be performed.

The *Process* for composition consists of the following steps [88]:

1. **Unification of the action sets of the individual patterns** In principle this step performs the union of the two sets of actions. However, it must be ensured that names of actions and parameters are unambiguous, i.e. distinguish those actions and parameters, respectively, that are equal but shall be different. Further, matching parts of the models, i.e. those action and parameter names that are different but shall be identical need to be identified and renamed. This task strongly depends on the actual description of the patterns.

2. **Refinement and linking of properties** The properties of the individual patterns usually only refer to all actions and parameters relevant for each particular property. If this is not sufficient for expressing the targeted combined property they have to be refined in order to include all necessary information. Combining properties can be achieved by linking particular parameters that are then included in the refined properties.

3. **Revision of the assumptions for the individual patterns** There are cases in which combinations of patterns result in weaker security just because some of the assumptions that hold for the system being based on one pattern are violated by the other pattern. Therefore, each assumption has to be checked in the context of the combined system.

4. **Functional integration and additional assumptions** The pattern integration on a functional level consists in a refinement of the unified actions which is rooted in the refinement of the properties. This integration may lead to additional assumptions that span over both integrated systems.

5. **Verification/validation of the integrated pattern** Validation or verification of the integrated solution is essential. Very often, intuitive solutions do not satisfy the expected security properties.
6. **Implementation** In the cases in which pattern integration includes more than just composition, it has to be expected that changes to the pattern implementation are required for the integrated pattern.

The paper uses an example to demonstrate the process where two solutions are to be integrated:

- **User Authentication through SSL** A SSL-Client-Authentication is used to authenticate the user that connects to a server, e.g. by using a SmartCard with the assumption that once a SSL-channel is authentically established, it will provide authenticity for the subsequent communication.
- **Device Identification** As specified by the Trusted Computing Group, a Trusted Platform Module (TPM) can be used to perform a TPM_Quote operation that attests the current configuration of a platform by producing a signature that can only be generated at a single TPM and thereby may identify the device to a server.

The properties provided by the respective solutions are then formalized as follows:

1. Whenever a server S performs an ssl-receive action refering to a client C_i, it is authentic for S that C_i established this channel.
2. Whenever a verifier V verifies an attestation refering to a device d_k, it is authentic for V that the TPM_Quote was produces on device d_k.

The property the pattern composition shall provide and which is derived by the step of *linking of properties* (see below) is then:

3. Whenever a server S has performed an ssl-receive action refering to a client C_i and has verified an attestation refering to a device d_k, it is authentic for S that C_i established this channel on device d_k and that the TPM_Quote was produces on the same device d_k by the same user C_i.

Applying the composition process to this example includes the following steps:

1. Since the server shall both use the SSL channel and receive and verify the quote message, the actor V of the device identification is renamed to S.
2. In order to being able to express on which device the SSL channel has been established by the client, the ssl-initialization action is enriched by the parameter d_l identifying the device it was performed on. Similarly, the TPM_Quote action is extended by the parameter C_j identifying the user who issued it, which concludes the step of *Refinement of properties*. The *linking of properties* is achieved by identifying the client and the channel parameters, respectively, used in the two property specifications, i.e. by setting $C_i = C_j$ and $d_k = d_l$. This formalizes that the device used for SSL channel establishment is the same that has been authentically identified by the

server ($d_k = d_l$) and that the user who initialized the TPM_Quote is the same that has established the SSL channel with the server ($C_i = C_j$) which results in property 3 above.

3. The *Revision of the assumptions for the individual patterns* step for the example does not require any adaptations. This step addresses the notions of *horizontal* and *vertical composability* of Mödersheim and Viganò.

4. While the previous steps can be considered more technical, the step of *Functional integration and additional assumptions* is clearly the task of a security expert, since there usually exist several different ways for it. As explained in [88], the above property can be achieved by inserting a new parameter in the quote message that identifies the SSL channel on which this message is sent. Three additional assumptions are presented: The authenticity of the SSL channel extends to the quote message, this message can only be sent when the SSL channel has been established, and the channel that is contained in the quote message is initiated on the device that generates this message. Other solutions with only a loose coupling of SSL channel and quote message do not provide the desired property as was shown in [90] (see Section 3.2.10).

5./6. The *Verification/validation of the integrated pattern* and the correct *Implementation* of the composition are then challenges that are external to the composition process introduced in [88] and may be performed through several different approaches.

This work presented in [88] demonstrates that it is not trivial to construct the correct composed properties and to achieve a composition that indeed provides these properties. The sheer number of steps required demonstrates the necessity for a systematic approach. This work however is only a first step towards such a systematic approach. In particular, the steps of property refinement and functional integration need to be further investigated. Further, the refinement and composition of systems and properties does not necessarily need to be based on system and property specification according to SeMF but seems to be generic and applicable to other modeling approaches. Scientific challenges include the adaption of the process to other approaches such as SeMF Security Building Blocks (so-called SeBBs), or attack-model based formalizations such as the one presented by Mödersheim and Viganò (the latter might require some adaptations for the process).

3.2.10 Verification through Abstraction

While the process described in the previous section includes the derivation of what is the goal of composing protocols, namely the “composition” of the properties separately provided by the protocols, it leaves open the task of verifying the correctness of the functional composition of the protocols, i.e. the verification of this composed property being actually fulfilled by the composed protocols. The verification approach that complements the generic process described above and addresses this task is presented by Fuchs, Gürgens and Rudolph in [90]. This approach utilizes the notion of security properties defined

within the Fraunhofer SIT Security Modelling Framework (SeMF) and so-called *Security Preserving Alphabetic Language Homomorphisms*.

The idea is to specify a concrete system that models the composition of solutions that shall provide a specific property, and to specify an abstract system that can easily be proven to provide this property. Then, a homomorphism is specified that maps the concrete system to the abstract one, and finally this homomorphism is proven to satisfy a specific, property preserving condition.

More specifically, system specification in SeMF consists in principle of the parts described below. However, these parts do not have to be specified in every detail and not all parts are needed for a specific system model. It is sufficient to model those parts of the system that are relevant for the specific properties that shall be addressed. Thus a system specification involves to specify

- the *agents* that act in the system and the *actions* they perform,
- *the behaviour* of the system that is composed of each of the agents' behaviours – describing what an agent will do in response to which kind of input,
- *the local views* of the agents of the system, defining which actions are visible to each of the agents – describing which actions an agent can see and therefore react on. (This may include that the agent sees only parts of an action, such as the message on a network bus, but not which agent put it there.)
- *the initial knowledge* of the agents that consist of which traces of actions an agent considers to be principally possible. (This also covers the learning curve of an agent as it may consider certain actions not possible after having observed a certain event.)

The approach introduced in [90] uses again the example of composing a user authentication based on SSL with a device identification based on TPM attestation. The specification of the abstract system explicitly excludes sequences of actions that violate the desired combined authenticity property, thus resulting in a trivial proof that the abstract system indeed provides this property. In a next step, a *concrete system* is formulated in terms of SeMF. This second system is completely independant of the idealized system and represents the actual solution for the functional composition of the protocols. This system is oriented directly towards the way that the protocols are integrated in a possible implementation. Again this system is not specified in every detail and only contains the relevant parts and assumptions.

Next, those two systems are related to each other. This is formally done with the use of an *alphabetic language homomorphism* from the concrete to the ideal system. A homomorphism describes a structure-preserving map between those two systems with the operation in both these systems being the concatenation, i.e. the successive execution of actions. Of course there usually are several possible homomorphisms and the security expert will have to choose the most promising for the next step.

The homomorphism constructed in [90] is then evaluated regarding its property to *preserve security* properties, in particular to preserve authenticity. For this Gürgens et al.

have defined according sets of sufficient conditions [91, 92]. The attempt to proof these conditions will then either result in a proof that the concrete system really fulfills the goals of the composition, or give hints as to where the composition of protocols may have weaknesses that open possibilities to attacks. The work in [90] gives an example for both a sufficient and an insufficient composition of the exemplary application for the composition verification. The fact that one of the two compositions fails to provide the desired composed authenticity property is revealed by the fact that the chosen homomorphism does not satisfy a specific condition necessary for preserving authenticity.

It has to be noted that this does not constitute a proof that the concrete system does not provide the property, it is more a hint for a security expert as to possible weaknesses of the composition. Further, sufficient conditions have so far been proven for authenticity, proof of authenticity, and confidentiality, hence the approach needs to be extended in order to cover other properties relevant for RCES.

3.3 Summary

A huge amount of work has already been performed, addressing different types of properties being preserved under composition. However, today's IT systems and in particular embedded systems are and will increasingly be used for many types of security relevant applications in different domains. They require a wide variaty of S&D properties to hold: authenticity, confidentiality, non-repudiation, trust properties, timeliness, to name a few. Hence the challenge is to develop methods and tools that address all these different properties and allow protocol composition with the aim of preserving S&D properties or even achieve stronger ones.

4 Formalization of Metrology Requirements

Nowadays type approval of a new metering device is a very time consuming process. Every new type of meter has to be approved individually which usually includes a type examination of a specimen of the new device. An approved design process for metering application software could help to relieve the admission offices. Our goal is to define such an approved trusted design process for metering application software. Using this trusted design process in development shall lead to trusted metering devices.

The definition of such a trusted design process involves several steps. At first the catalogues containing requirements on metering devices have to be identified. The catalogues need to be checked for requirements addressing security and dependability relevant aspects of legally relevant metering software. All these requirements from different catalogues need to be extracted and consolidated in subsequent steps. They also need to be checked for overlaps and discrepancies. Furthermore it's necessary to check for laws that cover the metering domain to make it possible to assure compliance with an approved design process. When the extraction of all metering requirements from catalogues and laws has been done, the research on potential formalization approaches can be started.

Regarding the extraction from requirement catalogues and lawful regulations which cover the metering sector, the focus will be on active electricity energy meters. Above all, requirements that cover security and dependability aspects related to legally relevant parts of the metering software (used in embedded systems smart meters) will be concentrated on. The smart meter sector is expected to be a growing market in the next years. §40 of the German ENWG states that every customer has the right to demand a monthly electricity bill. Smart meters will enable frequently billing by using communication techniques for automatic meter reading. The spreading of smart meters is also facilitated by the EU Directive 2006/32/EC [93]. This directive states that "*Member States shall ensure that [...] final customers [...] are provided with [...] meters that accurately reflect the final customer's actual energy consumption and that provide information on actual time of use*". Furthermore smart meters can help to support smart grid mechanisms that will be used to automatically fit the amount of produced power by distributed energy providers like block power stations to the actual demand.

4.1 Requirement Catalogues

In this section we will give insight into requirement catalogues that are relevant for the electricity metering sector. We will give an overview of the contents for each of the catalogues and the relevant requirements. We will also annotate for which countries the catalogue is

Annex	Meter type
MI-001	Water meters
MI-002	Gas meters and volume conversion devices
MI-003	Active electrical energy meters
MI-004	Heat meters
MI-005	Measuring systems for continuous and dynamic measurement of quantities of liquids other then water
MI-006	Automatic weighing instruments
MI-007	Taximeters
MI-008	Material measures
MI-009	Dimensional measuring instruments
MI-010	Exhaust gas analyzers

Table 4.1: Types of meters covered by the MID

relevant. The next research period will start with an extraction of the requirements covering security and dependability aspects related to legally relevant parts of the metering software. And an investigation if the contents of the catalogues overlap.

4.1.1 MID

The MID (Measuring Instruments Directive, [94]) is an EU directive that has been published 2004-05-31. It had to be transferred to national law in each of the EU member countries until 2006-01-04 and should be applied since 2006-10-30. The contents of the MID cover various types of devices and systems with a measuring function. The document is structured into a main part consisting of 27 articles describing how this directive has to be applied and several annexes which actually contain the requirements to the metering devices. Annex I contains requirements that apply to all classes of metering devices. The special requirements to the different classes of metering devices can be found in annexes MI-001 to MI-010. The remaining annexes A to H1 contain information about different conformity assessment procedures. Table 4.1 gives an overview about the types of meters covered by the MID.

Regarding active electricity energy meters, the MI-003 replaces the previous appropriate directive 76/891/EEC [95]. Most of the requirements in annex I and MI-003 cover topics that aren't related to the software on active electrical energy meters. For example topics like mechanical properties, fault tolerance ranges or even information and markings that have to be put on the case of the instrument are described.

The following requirements of the MID are the ones that are related to security and reliability of the legally relevant parts of metering software:

Annex I

7. *Suitability*

7.2 A measuring instrument shall have no feature likely to facilitate fraudulent use, whereas possibilities for unintentional misuse shall be minimal.

7.6 A measuring instrument shall be designed so as to allow the control of the measuring tasks after the instrument has been placed on the market and put into use. If necessary, special equipment or software for this control shall be part of the instrument. The test procedure shall be described in the operation manual.

When a measuring instrument has associated software which provides other functions besides the measuring function, the software that is critical for the metrological characteristics shall be identifiable and shall not be inadmissibly influenced by the associated software.

8. *Protection against corruption*

8.1 The metrological characteristics of a measuring instrument shall not be influenced in any inadmissible way by the connection to it of another device, by any feature of the connected device itself or by any remote device that communicates with the measuring instrument.

8.2 A hardware component that is critical for metrological characteristics shall be designed so that it can be secured. Security measures foreseen shall provide for evidence of an intervention.

8.3 Software that is critical for metrological characteristics shall be identified as such and shall be secured.

Software identification shall be easily provided by the measuring instrument. Evidence of an intervention shall be available for a reasonable period of time.

8.4 Measurement data, software that is critical for measurement characteristics and metrologically important parameters stored or transmitted shall be adequately protected against accidental or intentional corruption.

8.5 For utility measuring instruments the display of the total quantity supplied or the displays from which the total quantity supplied can be derived, whole or partial reference to which is the basis for payment, shall not be able to be reset during use.

10. *Indication of result*

10.5 Whether or not a measuring instrument intended for utility measurement purposes can be remotely read it shall in any case be fitted with a metrologically controlled display accessible without tools to the consumer. The reading of this display is the measurement result that serves as the basis for the price to pay.

Annex MI-003

5. *Suitability*

5.2. The display of the total energy shall have a sufficient number of digits to ensure that when the meter is operated for 4 000 hours at full load ($I = I_{max}$, $U = U_n$ and $PF = 1$) the indication does not return to its initial value and shall not be able to be reset during use.

5.3. In the event of loss of electricity in the circuit, the amounts of electrical energy measured shall remain available for reading during a period of at least 4 months.

Annex MI-003 also states that the manufacturer of a new active electrical meter can choose between the following conformity assessment procedures: B+F or B+D or H1. An overview of the different options is given in chapter 4.3.

The regulations of the MID apply to metering devices until they are placed in the market.

4.1.2 WELMEC

The WELMEC 7.2 Software Guide [96] is a document that has been written by the WG7 (working group 7: software) of the WELMEC (European Cooperation in Legal Metrology). The WELMEC was founded on 8. June 1990 with 13 representative Members from national authorities responsible for legal metrology in European Union and EFTA Member States. Currently there are 30 Members and 7 Associate Members in the WELMEC Committee. Detailed information about the member states can be found at [97].

The intention of the WELMEC 7.2 Software Guide is to provide advice to developers of metering software how the requirements of the MID can be fulfilled. These guidelines are only examples that fulfill the MID requirements, but they aren't mandatory. On the other hand side it can speed up the type approval of a new device, when the well known solutions described in the WELMEC 7.2 have been used. The current version of the document is issue 4 and has been published at May 2009.

The document is structured in two chapters which contain the basic requirements and fife chapters which contain the extensions L, T, S, D and I. The basic requirements describe two types of systems. Type P systems are built for purpose measuring instruments, for example instruments based on embedded systems. Type U systems are universal computers. Active electricity meters are usually devices of type P, thus the chapter about type U systems is not taken into account. The extensions are:

- Extension L: Long-term Storage of Measurement Data
- Extension T: Transmission of Measurement Data via Communication Networks
- Extension S: Software Separation
- Extension D: Download of Legally Relevant Software
- Extension I: Instrument Specific Software Requirements

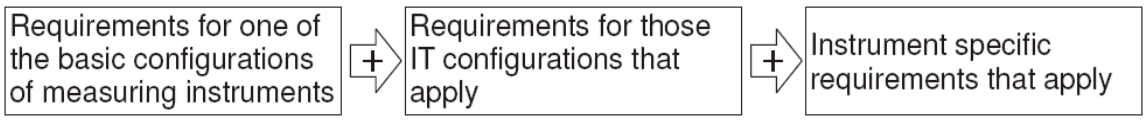

Figure 4.1: Combination of requirement sets for a concrete metering device.

The extensions L, T, S and D describe so-called IT configurations. Extension I covers the requirements to specific metering devices Water Meters, Gas Meters and Volume Conversion Devices, Active Electrical Energy Meters and Heat Meters. We will focus on requirement set I3 which describes Active Electrical Energy Meters. Figure 4.1 shows how the chapters should be combined for a specific metering device. Figure 4.2 gives an overview of the whole document.

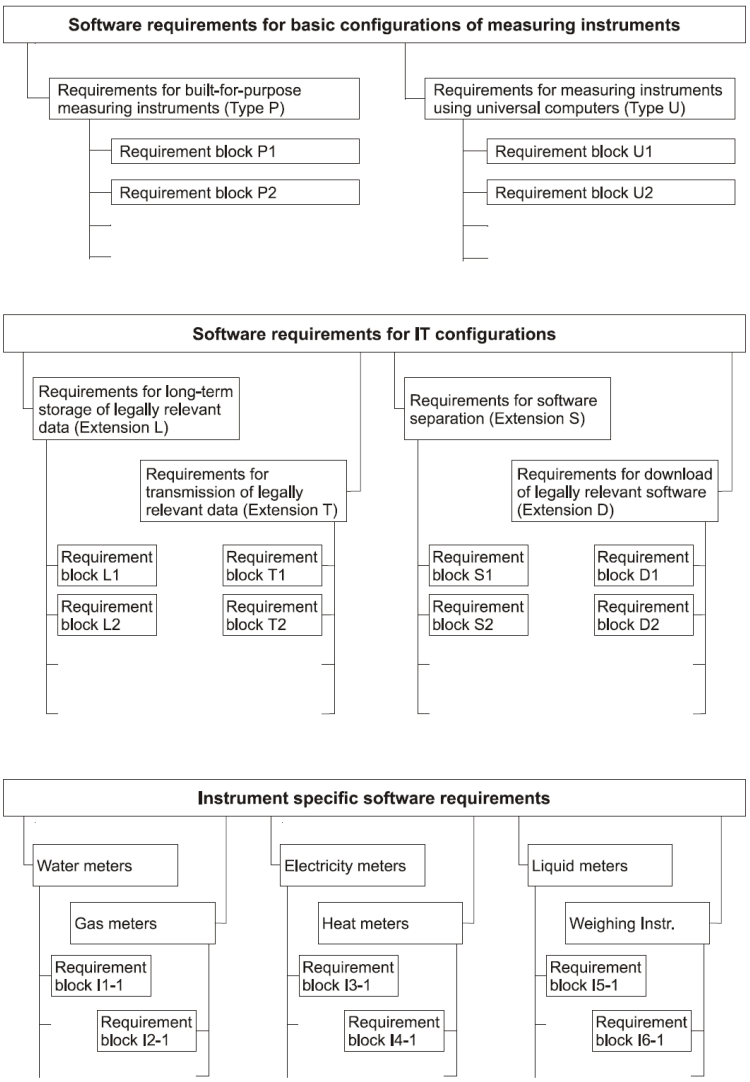

Figure 4.2: Overview of the WELMEC 7.2 requirement sets.

Any of the requirement blocks in the WELMEC 7.2 contain the following information:

- The requirement itself with optional additional specifying notes.
- A list of required documentation for the conformity assessment.
- A validation guidance describing which checks are performed by the notified body to validate if the requirement is fulfilled.

- An example of an acceptable solution.

The requirement blocks also consider the risk classes B, C and D for each requirement. The exact definition of the risk classes can be found in chapter 11 of the WELMEC 7.2. Active electricity meters are assigned to risk class C in the WELMEC 7.2, although in Germany it's a national requirement that meters capable to transmit data over open networks are assigned to risk class D.

The following requirement blocks are related to security and dependability in the legally relevant part of active electricity meter software.

P3: Influence via user interface
Commands entered via the user interface shall not inadmissibly influence the legally relevant software and measurement data.

P4: Influence via communication interface
Commands inputted via communication interfaces of the instrument shall not inadmissibly influence the legally relevant software and measurement data.

P5: Protection against accidental or unintentional changes
Legally relevant software and measurement data shall be protected against accidental or unintentional changes.

P6: Protection against intentional changes
Legally relevant software shall be secured against the inadmissible modification, loading or swapping of hardware memory.

P7: Parameter protection
Parameters that fix legally relevant characteristics of the measuring instrument shall be secured against unauthorized modification.

L1: Completeness of measurement data stored
The measurement data stored must contain all relevant information necessary to reconstruct an earlier measurement.

L2: Protection against accidental or unintentional changes
Stored data shall be protected against accidental and unintentional changes.

L3: Integrity of data
The measurement data stored must be protected against intentional changes.

L4: Authenticity of measurement data stored
Authenticity of measurement data stored must be capable of being authentically traced back to the measurement that generated them.

L5: Confidentiality of keys
Keys and accompanying data must be treated as legally relevant data and must be kept secret and be protected against compromise by software tools.

L6: Retrieval of stored data
The software used for verifying measurement data sets stored shall display or print the data, check the data for changes, and warn if a change has occurred. Data that are detected as having been corrupted must not be used

L7: Automatic storing
The measurement data must be stored automatically when the measurement is concluded.

L8: Storage capacity and conformity
The long-term storage must have a capacity which is sufficient for the intended purpose.

T1: Completeness of transmitted data
The transmitted data must contain all relevant information necessary to present or further process the measurement result in the receiving unit.

T2: Protection against accidental or unintended changes
Transmitted data shall be protected against accidential and unintentional changes.

T3: Integrity of data
The legally relevant transmitted data must be protected against intentional changes with software tools.

T4: Authenticity of transmitted data
For the receiving program of transmitted relevant data, it shall be possible to verify the authenticity and the assignment values to a certain measurement.

T5: Confidentiality of keys
Keys and accompanying data must be treated as legally relevant data and must be kept secret and be protected against compromise by software tools.

T6: Handling of corrupted data
Data that are detected as having been corrupted must not be used.

T7: Transmission delay
The measurement must not be inadmissibly influenced by a transmission delay.

T8: Availability of transmission services
If network services become unavailable, no measurement data must get lost.

S3: Protective software interface
The data exchange between the legally relevant and non-legally relevant software must be performed via a protective software interface, which comprises the interactions and data flow.

D1: Download mechanism
Downloading and the subsequent installation of software shall be automatic and shall ensure that the software protection environment is at the approved level of completion

D2: Authentication of downloaded software
Means shall be employed to guarantee that the downloaded software is authentic, and to indicate that the downloaded software has been approved by an NB.

D3: Integrity of downloaded software
Means shall be employed to guarantee that the downloaded software has not been inadmissibly changed during download

D4: Traceability of legally relevant software download
It shall be guaranteed by appropriate technical means that download of legally relevant software are adequately traceable within the instrument for subsequent controls.

I3-1: Fault Recovery
The Software shall recover from a disturbance to normal processing

I3-2: Back-up Facilities
There shall be a facility that provides for the periodic back-up of legally relevant data, such as measurement values, and the current status of the process in case of a disturbance. This data shall be stored in non-volatile storage

I3-4: MID-Annex I, 8.5 (Inhibit resetting of cumulative measurement values)
For utility measuring instruments the display of the total quantity supplied or the displays from which the total quantity supplied can be derived, whole or partial reference to which is the basis for payment, shall not be able to be reset during use.

I3-5: Dynamic behaviour
The non-legally relevant software shall not adversely influence the dynamic behavior of a measuring process.

4.1.3 PTB-A 50.7

The PTB-A 50.7 [98] is a technical guideline published by the German institute PTB (Physikalisch Technische Bundesanstalt / Physical Technical Federal Institute) in April 2002. Its requirements are similar to those of normative documents, but they aren't legally relevant. Like the WELMEC 7.2, the PTB-A 50.7 is just a guide that should help to speed up the process of type approval. The contents of the PTB-A 50.7 cover metering devices with complex additional features that are not specified in the MID. The PTB-A 50.7 doesn't conflict with the MID requirements.

The requirements on software are specified in the annexes PTB-A 50.7-1, PTB-A 50.7-2 and PTB- A50.7-3. Any of these annexes cover different classes of metering devices, as listed in table 4.2.

Annex	Translated title	Original title
PTB-A 50.7-1	Device class 1: Simple device	Geräteklasse 1: Einfaches Gerät
PTB-A 50.7-2	Device class 2: Device using data transmission over communication networks	Geräteklasse 2: Gerät mit Datenübertragung über Kommunikationsnetzwerke
PTB-A 50.7-3	Device class 3: Device with software separation	Geräteklasse 3: Gerät mit Software-Trennung

Table 4.2: Device classes covered by the PTB-A 50.7

4.1.4 OIML

The Organisation Internationale de Métrologie Légale is a worldwide, intergovernmental organization which was founded October 12th 1955 in Paris.

Its aim is the regulation of measurement needs regarding legal metrology. The OIML provides documents which are admitted in many countries and often integrated into engineering standards.

The OIML provides four main categories of documents:

- **International Recommendations (OIML R)**, which are model regulations that establish the metrological characteristics required of certain measuring instruments and which specify methods and equipment for checking their conformity. OIML Member States shall implement these Recommendations to the greatest possible extent;
- **International Documents (OIML D)**, which are informative in nature and which are intended harmonize and improve work in the field of legal metrology;
- **International Guides (OIML G)**, which are also informative in nature and which are intended to give guidelines for the application of certain requirements to legal metrology;
- **International Basic Publications (OIML B)**, which define the operating rules of the various OIML structures and systems.

Concerning security and dependability of embedded systems in the metering sector, the document "General requirements for software controlled measuring instruments" is of great interest. It was developed by the OIML Technical Subcommittee TC 5/SC 2 Software and published as international document OIML D 31 (Latest issue: 2008, [99]).

The information and instructions given in this document apply only to software controlled measuring instruments as for example embedded systems. It comprises requirements for the legally relevant software part. The OIML document contains the following requirements:

General requirements (5.1):

Software identification (5.1.1)

Legally relevant software of a measuring instrument / electronic device / sub assembly shall be clearly identified with the software version or another token. The identification may consist of more than one part but at least one part shall be dedicated to the legal purpose.

Correctness of algorithms and functions (5.1.2)

The measuring algorithms and functions of an electronic device shall be appropriate and functionally correct for the given application and device type (accuracy of the algorithms, price calculation according to certain rules, rounding algorithms, etc.).

Software protection (5.1.3)

- **Prevention of misuse (5.1.3.1):** A measuring instrument shall be constructed in such a way that possibilities for unintentional, accidental, or intentional misuse are minimal. In the framework of this OIML Document, this applies especially to the software. The presentation of the measurement results should be unambiguous for all parties affected.
- **Fraud protection (5.1.3.2):** The legally relevant software shall be secured against unauthorized modification, loading, or changes by swapping the memory device. In addition to mechanical sealing, technical means may be necessary to secure measuring instruments having an operating system or an option to load software..
- Only clearly documented functions (...) are allowed to be activated by the user interface, which shall be realized in such a way that it does not facilitate fraudulent use.
- Parameters that fix the legally relevant characteristics of the measuring instrument shall be secured against unauthorized modification. If necessary for the purpose of verification, the current parameter settings shall be able to be displayed or printed.
- Software protection comprises appropriate sealing by mechanical, electronic and/orcryptographic means, making an unauthorized intervention impossible or evident.

Requirements specific for configurations (5.2):

Specifying and separating relevant parts and specifying interfaces of parts (5.2.1)

Metrologically critical parts of a measuring system – whether software or hardware parts – shall not be inadmissibly influenced by other parts of the measuring system.

Separation of electronic devices and subassemblies (5.2.1.1)

- Subassemblies or electronic devices of a measuring system that perform legally relevant functions shall be identified, clearly defined, and documented. They form the legally relevant part of the measuring system.

- During type testing, it shall be demonstrated that the relevant functions and data of subassemblies and electronic devices cannot be inadmissibly influenced by commands received via the interface.

Separation of software parts (5.2.1.2)

- All software modules (programs, subroutines, objects, etc.) that perform legally relevant functions or that contain legally relevant data domains form the legally relevant software part of a measuring instrument (electronic device or sub assembly).
- If the legally relevant software part communicates with other software parts, a software interface shall be defined. All communication shall be performed exclusively via this interface. The legally relevant software part and the interface shall be clearly documented. All legally relevant functions and data domains of the software shall be described to enable a type approval authority to decide on correct software separation.
- There shall be an unambiguous assignment of each command to all initiated functions or data changes in the legally relevant part of the software. Commands that communicate through the software interface shall be declared and documented. Only documented commands are allowed to be activated through the software interface. The manufacturer shall state the completeness of the documentation of commands.
- Where legally relevant software has been separated from nonrelevant software, the legally relevant software shall have priority using the resources over nonrelevant software. The measurement task (realized by the legally relevant software part) must not be delayed or blocked by other tasks.

Shared indications (5.2.2)

A display or printout may be employed for presenting both information from the legally relevant part of software and other information. The contents and layout are specific for the kind of instrument and area of application and have to be defined in the relevant Recommendation.

Storage of data, transmission via communication systems (5.2.3)

- If measurement values are used at another place than the place of measurement or at a later time than the time of measurement they possibly have to leave the measuring instrument (electronic device, subassembly) and be stored or transmitted in an insecure environment before they are used for legal purposes. In this case the following requirements apply:
- The measurement value stored or transmitted shall be accompanied by all relevant information necessary for future legally relevant use. (5.2.3.1)
- The data shall be protected by software means to guarantee the authenticity, integrity and, if necessary correctness of the information concerning the time of measurement. The software that displays or further processes the measurement values and accompanying data shall check the time of measurement,

authenticity, and integrity of the data after having read them from the insecure storage or after having received them from an insecure transmission channel. If an irregularity is detected, the data shall be discarded or marked unusable. Software modules that prepare data for storing or sending, or that check data after reading or receiving, belong to the legally relevant software part. (5.2.3.2)

- For a high protection level it is necessary to apply cryptographic methods. Confidential keys employed for this purpose shall be kept secret and secured in the measuring instruments, electronic devices, or subassemblies involved. Means shall be provided whereby these keys can only be input or read if a seal is broken. (5.2.3.3)

Automatic storing (5.2.3.4)

- When, considering the application, data storage is required, measurement data must be stored automatically when the measurement is concluded, i.e. when the final value used for the legal purpose has been generated. The storage device must have sufficient permanency to ensure that the data are not corrupted under normal storage conditions. There shall be sufficient memory storage for any particular application. When the final value used for the legal purpose results from a calculation, all data that are necessary for the calculation must be automatically stored with the final value.
- Stored data may be deleted if either the transaction is settled orthese data are printed by a printing device subject to legal control.
- (...) when the storage is full, it is permitted to delete memorized data when both of the following conditions are met. Data are deleted in the same order as the recording order and the rules established for the particular application are respected and deletion is carried out either automatically or after a special manual operation.

Transmission delay (5.2.3.5)

The measurement shall not be inadmissibly influenced by a transmission delay.

Transmission interruption (5.2.3.6)

If network services become unavailable, no measurement data shall be lost. The measurement process should be stopped to avoid the loss of measurement data.

Time stamp (5.2.3.7)

The time stamp shall be read from the clock of the device. Depending on the kind of instrument, or area of application, setting the clock may be legally relevant and appropriate protection means shall be taken according to the severity level to be applied. The internal clock of a standalone measuring instrument tends to have a large uncertainty because there is no means to synchronize it with the global clock. But if the information concerning the time of measurement is necessary for a specific field of application, the reliability of the internal clock of the measuring instrument shall be enhanced by specific means.

Compatibility of operating system and hardware, portability (5.2.4)

- The manufacturer shall identify the hardware and software environment that is suitable. Minimum resources and a suitable configuration which is necessary for correct functioning shall be declared by the manufacturer. (5.2.4.1)
- Technical means shall be provided to prevent operation, if the minimal configuration requirements are not met (5.2.4.2).

Maintenance and reconfiguration (5.2.6)

Updating the legally relevant software of a measuring instrument in the field should be considered as a modification of the measuring instrument, when exchanging the software with another approved version or a repair of the measuring instrument, when reinstalling the same version. A measuring instrument which has been modified or repaired while in service may require initial or subsequent verification, dependant on national regulations. Software which is not necessary for the correct functioning of the measuring instrument does not require verification after being updated.

Verified Update (5.2.6.2)

After the update of the legally relevant software of a measuring instrument (exchange with another approved version or reinstallation) the measuring instrument is not allowed to be employed for legal purposes before a verification of the instrument has been performed and the securing means have been renewed.

Traced Update (5.2.6.3)

- Traced Update of software shall be automatic. On completion of the update procedure thesoftware protection environment shall be at the same level as required by the type approval.
- The target measuring instrument shall have fixed legally relevant software.
- Technical means shall be employed to guarantee the authenticity of the loaded software. If the loaded software fails the authenticity check, the instrument shall discard it and use the previous version of the software or switch to an inoperable mode.
- Technical means shall be employed to ensure the integrity of the loaded software, i.e. that it has not been inadmissibly changed before loading.
- Appropriate technical means shall be employed to ensure that Traced Updates are adequately traceable within the instrument.
- The measuring instrument shall have a subassembly / an electronic device for the user or owner to express his consent. It shall be possible to enable and disable this subassembly / electronic device e.g. by a switch that can be sealed or by a parameter. If the subassembly / electronic device is enabled, each download has to be initiated by the user or owner. If it is disabled no activity by the user or owner is necessary to perform a download.

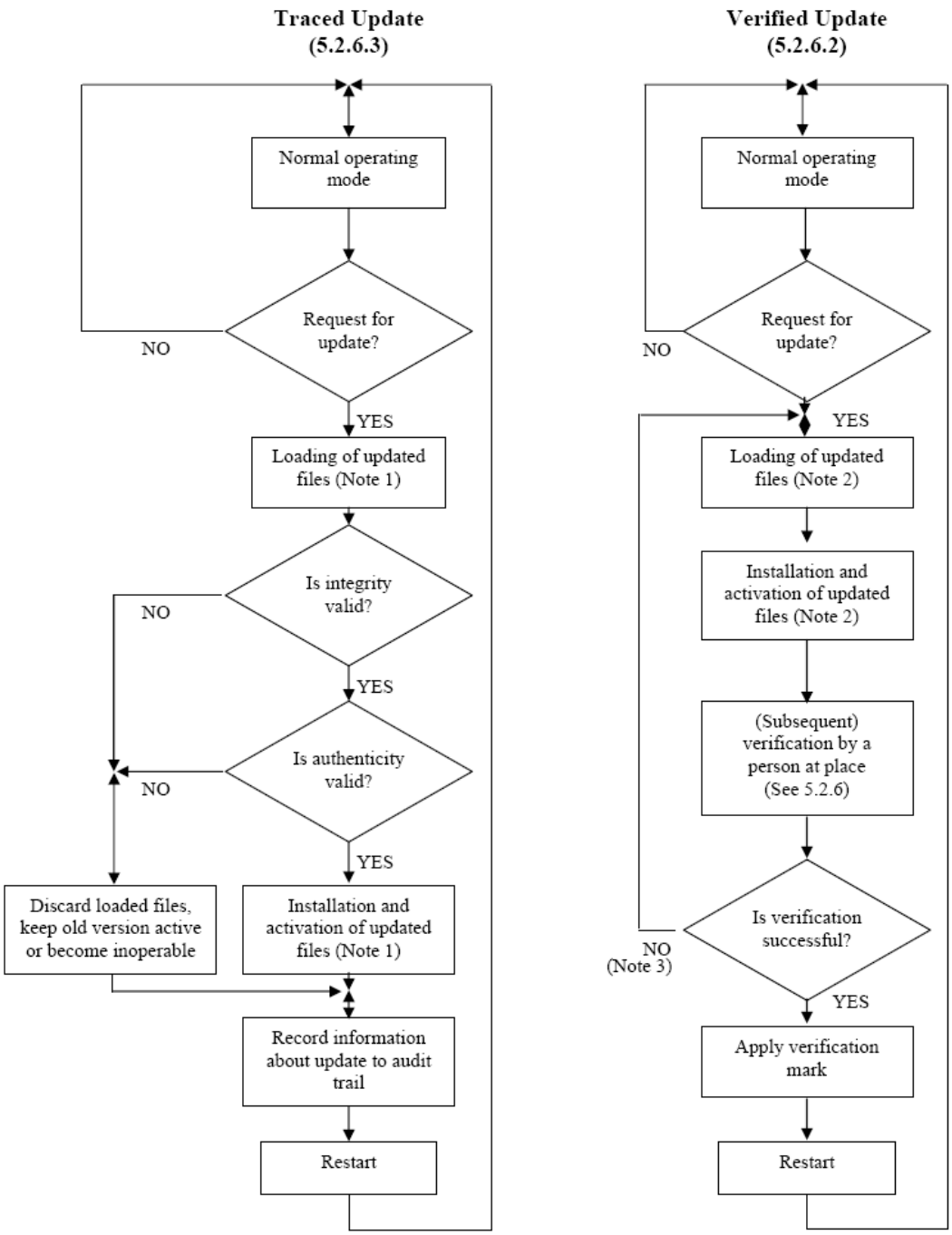

Figure 4.3: Traced Update and Verified Update.

- If the requirements 5.2.6.3.a through 5.2.6.3.f cannot be fulfilled, it is still possible to update the legally nonrelevant software part. In this case the following requirements shall be met: There is a distinct separation between the legally relevant and nonrelevant software according to 5.2.1.The whole legally relevant

software part cannot be updated without breaking a seal; it is stated in the type approval certificate that updating of the legally nonrelevant part is acceptable.

- The measuring instrument shall be fitted with a facility to automatically and non erasably record any adjustment of the device specific parameter, e.g. an audit trail. The instrument shall be capable of presenting the recorded data. (5.2.6.4)
- The traceability means and records are part of the legally relevant software and should be protected as such. (5.2.6.5)

4.1.5 Relationship of the catalogues

The above requirement catalogues aren't free of influences to each other. Figure 4.4 shows the chronological relations of the catalogues.

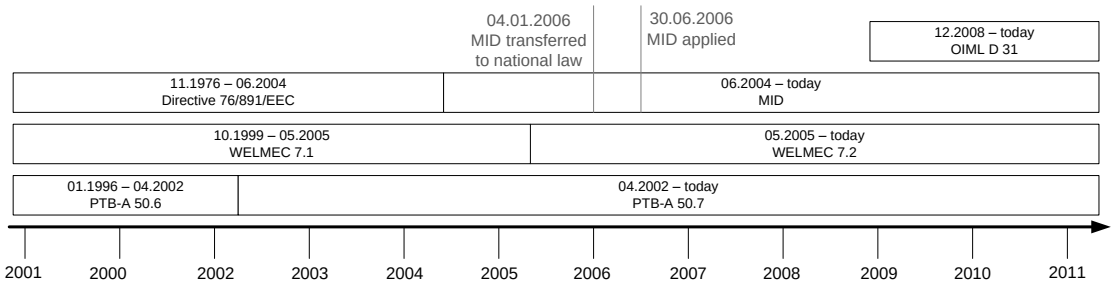

Figure 4.4: Chronology of the metrology catalogues.

The PTB-A 50.7 is a guideline that has been published by the German institute PTB (Physikalisch Technische Bundesanstalt / Physical Technical Federal Institute) in April 2002. It is valid for Germany but in the past other European countries used the guides from the PTB for their type approval process as well. The software requirements of the PTB-A 50.7 are derived from requirements of the WELMEC 7.1 and do not conflict with the MID.

The WELMEC 7.1 has been superseded by the WELMEC 7.2 Software Guide. This document has been published by the WG7 (working group 7: software) of the WELMEC (European Cooperation in Legal Metrology) at May 2005. It is valid in large parts of Europe, 37 European states are (associate) members in the WELMEC Committee. The contents of the WELMEC 7.2 Software Guide are guidelines how to fulfill the requirements of the MID.

The MID is an EU directive that has been published 2004-05-31. Its requirements had to be transferred to national law in all EU member states until 2006-01-04 and have to be applied since 2006-06-30.

The OIML is a worldwide, intergovernmental organization. The OIML D 31 contains recommendations that are referenced to in many national laws.

4.2 Lawful requirements in Germany

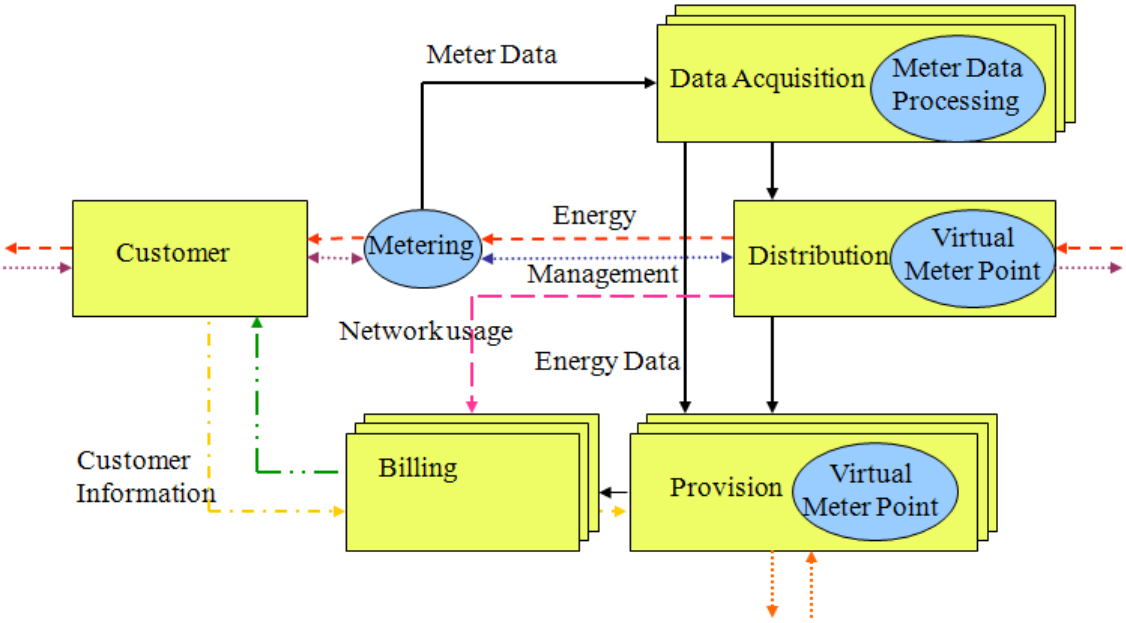

Figure 4.5: Metering domain overview.

Regarding the metering sector, different lawful requirements have to be considered. They cover different domains, from data acquisition until billing. Picture 4.5 gives an overview how the different paths of information data and energy are managed. The center of the picture shows a metering device. It stands for the basic sensor providing the current measurement information. The "Data Acquisition" field is the legally relevant part of the metering device, converting the raw sensor values into measurement data which is directly protected with use of digital signatures.

4.2.1 Energy Economy Law

Since 1st of January 2010 for new buildings and major renovations electronic meters for electricity and gas (so-called smart meters) have to be installed if this is technically feasible and economically reasonable. The energy suppliers have to offer those meters for existing buildings as well. The energy economy law is the German national outworking of the Energy end-use efficiency and energy services, directive (2006/32/EC) the main purpose of the law is:

- Establishing indicative targets, incentives and the institutional, financial and legal frameworks needed to eliminate market barriers and imperfections which prevent efficient end use of energy

- Creating the conditions for the development and promotion of a market for energy services and for the delivery of energysaving programmes and other measures aimed at improving enduse energy efficiency

Furthermore it regulates the liberalized energy market for example by formulating the primary care duty with requirements to the energy suppliers.

4.2.2 Digital Signature Act (SiG)

The Digital Signature Act [100] was enacted as Article 3 of the German Information and Communication Services Act. The purpose the Act is to establish general conditions under which digital signatures are considered as secure and counterfeits of digital signatures can reliably be proven. This makes them equivalent to ordinary signatures

In terms of metering devices, the Digital Signature Act is applicable for the creation of signatures for the measurement values after their acquisition to ensure authentication of origin and integrity of the data.

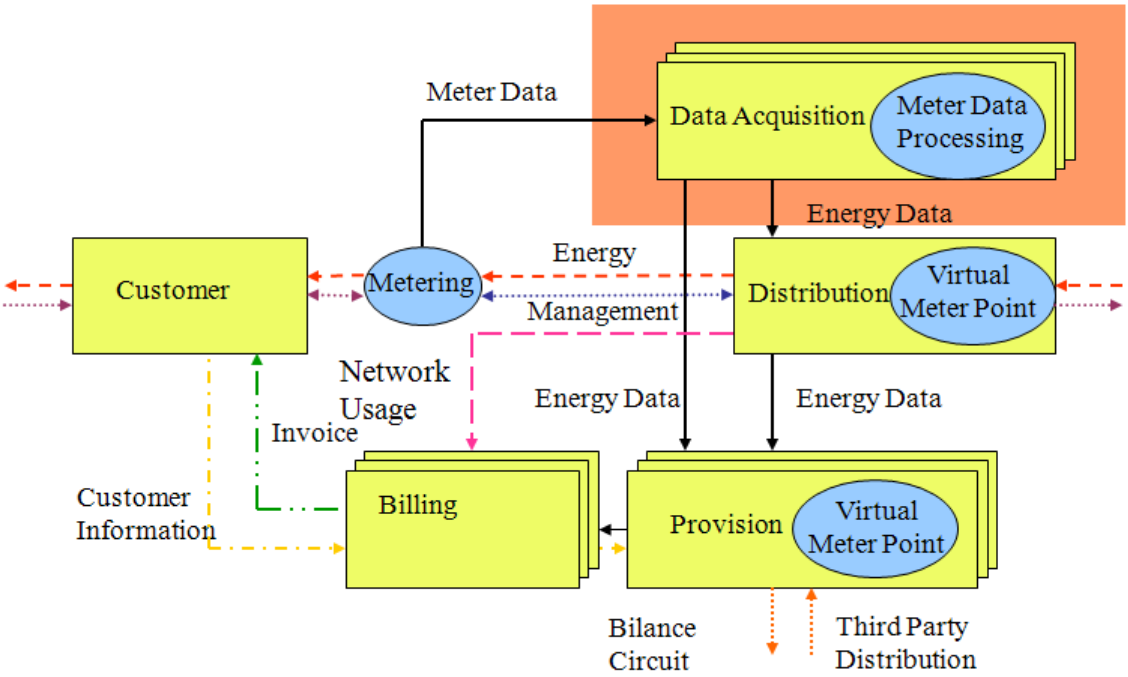

Figure 4.6: Domain regulated by the Digital Signature Act.

The law covers the domain of the metering device itself, where digital signatures are computed. By protecting them against manipulation the transparency of the data from origin (data acquisition) till billing for all market participants, as demanded by the Energy Economy Law, is ensured.

4.2.3 Verification Act (EichG)

The primary version of the national German Verification Act [101] was introduced on July, 11th 1969 and the actual version is from July 2008.

The main purpose of the law is to:

- Protect the consumer in terms of goods and services purchase and create the pre-condition for accurate measuring in course of business.
- Guarantee the measurement reliability in the area of health protection,, employment protection and environment protection.
- Strengthen trust in official/ministerial measuring.

Furthermore 23 appendices specify details of the special lawful regulations and the maximum permissible error rates for the individual measuring instrument categories. The appliance and implementation is the duty of the federal states of Germany.

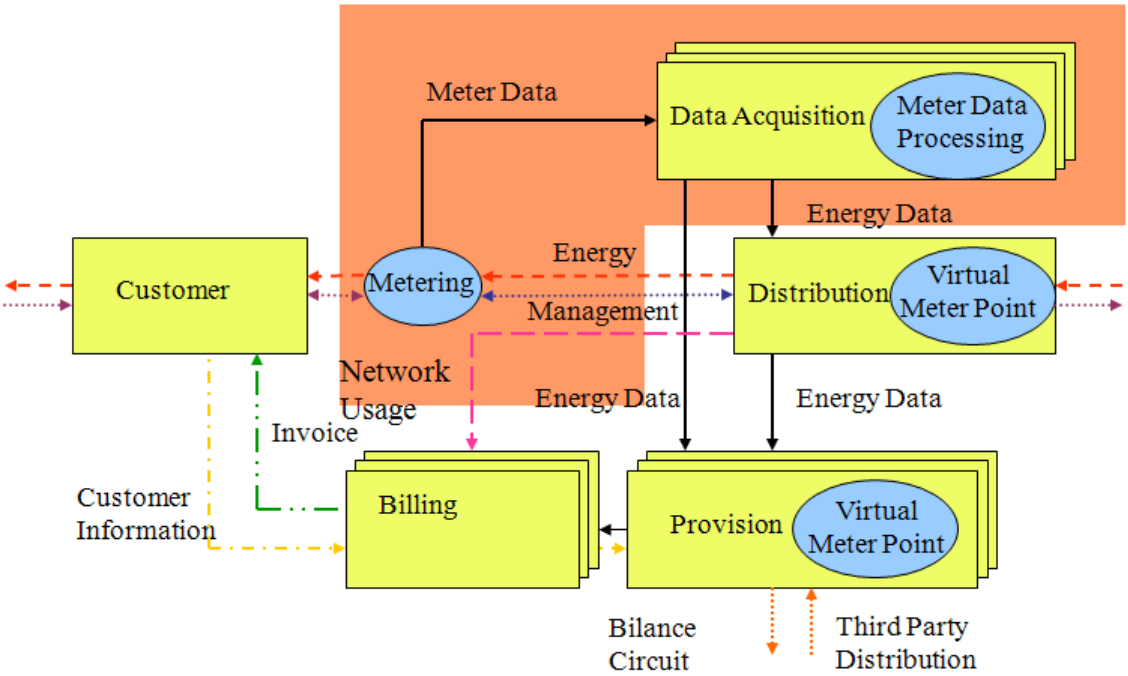

Figure 4.7: Domain regulated by the Verification Act.

In contrast to the MID, the German Verification Act has validity for all metering devices after they have been placed to market. It covers the legally relevant part of the meter, the sensor and the data acquisition part.

4.2.4 Federal Data Protection Act (BDSG)

The German Federal Data Protection Act [102] is the implementation of directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data.

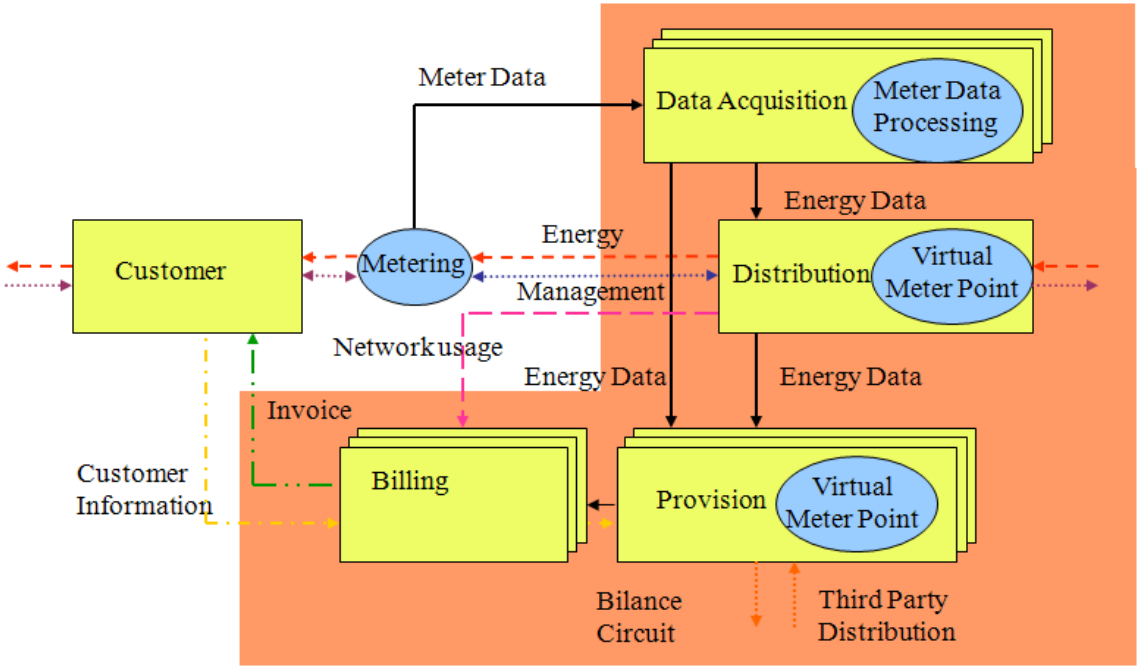

Figure 4.8: Domain regulated by the Federal Data Protection Act.

The purpose is to protect the individual against his right to privacy being impaired through the handling of his personal data.

The law covers the domain from data acquisition over distribution, provision until billing. It is to ensure privacy for the consumers in terms of measurement values and energy consumption profiles.

4.2.5 Telecommunications Data Protection Ordinance (TDSV), Telecommunications Act (TKG) , Teleservices Data Protection Act (TDDSG)

The three acts TDSV, TKG [103] and TDDSG [104] cover the domain shown in picture 4.9. The Telecommunications Data Protection Ordinance is of great importance for the electronic commerce. This ordinance regulates the protection of the personal data of parties engaging in telecommunications in respect of the collection, processing and use of these data by companies and persons providing telecommunications services on a commercial basis or contributing to such provision.

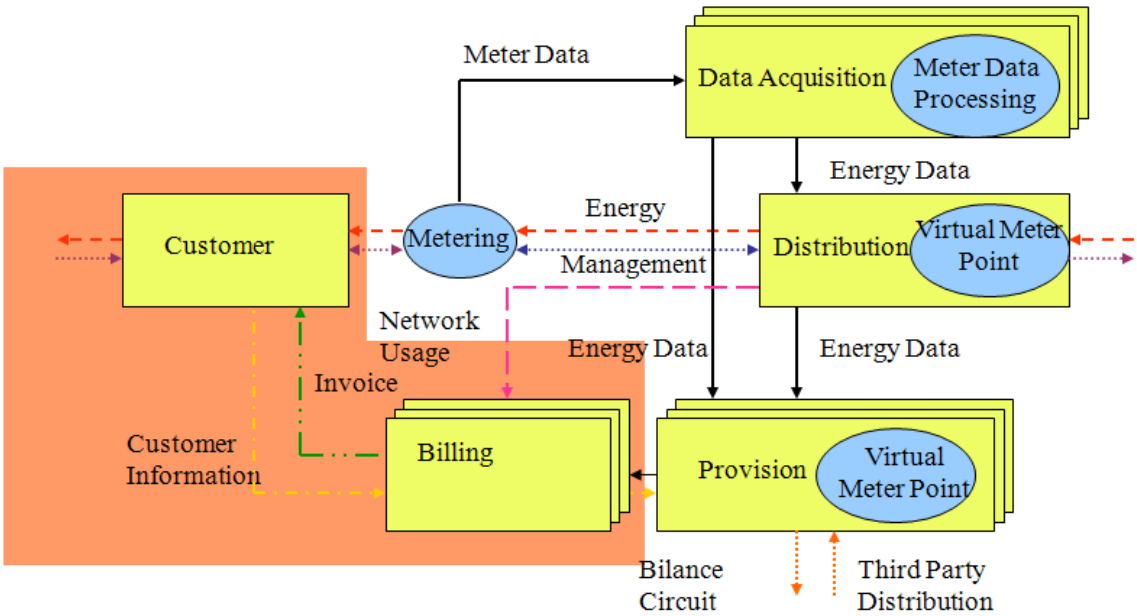

Figure 4.9: Domain regulated by the Telecommunications Data Protection Ordinance.

Unless otherwise provided in the Ordinance or in other special legislation, the provisions of the Federal Data Protection Act apply.

4.3 Type approval today

The type approval of metering devices is specified in the MID. The annexes A to H1 contain all possible conformity assessment procedures for any kind of metering device. For manufacturers of active electricity meters it's possible to chose between three combinations of conformity assessment procedures. These are the ones described in annexes B + D, B + F or H1. Subsequently a short overview of these procedures will be provided.

Annex B of the MID describes the conformity assessment procedure "Type Examination". This has to be combined with one of the procedures in annex D or annex F to achieve the type approval. The Type Examination is carried out by a Notified Body based on one of the following three options:

1. A specimen of the complete measuring instrument.
2. Specimens of one or more critical parts of the measuring instrument plus technical documentation that assesses the adequacy of the technical design of the other parts.
3. Assessment of the adequacy of the technical design by providing technical documentation.

The notified body will examine the technical documentation and the specimens to check if all requirements are fulfilled.

Annex D of the MID describes the "Declaration of conformity to type based on quality assurance of the production process". In that case the manufacturer has to provide documentation about the quality system, like quality objectives of the organization structure, qualification reports of the personnel concerned, inspection reports and calibration data. Based on this documentation, the notified body will judge if the quality system is sufficient. The "Declaration of conformity to type based on quality assurance of the production process" has to be combined with a "Type Examination" (annex B) to gain type approval.

Annex F describes the "Declaration of conformity to type based on product verification". This may happen in two different ways:

- All instruments are examined individually.
- Random samples from each lot produced are examined to determine if the lot is accepted or rejected.

The "Declaration of conformity to type based on product verification" has to be combined with a "Type Examination" (annex B) to gain type approval.

The manufacturer of an active electricity meter can choose between the conformity assessment procedures described in Annex D and Annex F, but either one has to be combined with the procedure described in Annex B. Another alternative is the process described in Annex H1 which doesn't need to be combined with the process described in Annex B.

Annex H1 describes the "Declaration of conformity based on full quality assurance plus design examination". This conformity assessment procedure may be carried out without the need for a "Type Examination". Similar to the "Declaration of conformity to type based on quality assurance of the production process" (annex D), the quality system of the manufacturer is assessed. Compared to the assessment described in annex D, not only the quality system concerning product quality is considered but also the quality system concerning design quality. The assessment includes design control and design verification techniques, processes and systematic actions. Additionally the design documentation of the new metering device must be made available to the Notified Body in charge.

This conformity assessment procedure is rarely used today but it could become more interesting to meter manufacturers when the results of the TERESA project become available. If there would be a well-known, trusted design process, that maybe even is approved by a metrology institute, the conformity assessment procedure descibed in annex H1 of the MID could become very interesting.

This conformity assessment procedure is rarely used today but it could become more interesting to meter manufacturers when the results of the TERESA project become available. Then the software design for a new metering device can be performed using trusted S&D patterns. Another benefit of S&D patterns for metering software development is a speed up of the assessment for the design. Using a trusted and approved design process, which integrates the needed S&D patterns would even be more efficient.

4.4 State of the art in metering development

Traditionally most meter developers are hardware experts. In the past most meters were pure mechanical or electromechanical constructions, like for example the Ferraris meters that use eddy current to drive a disc to measure the consumed power.

But the development progress in the metering sector led to built-for-purpose smart meters, which are usually based on embedded systems. As embedded systems run software, companies developing meters are nowadays faced with software engineering. Due to that fact, the formerly by electrical engineers dominated metering domain now has a demand for software engineers and appropriate engineering processes for metering software development.

But as far as most meter developers are hardware experts they do not know much about software engineering processes and even less about security and dependability. Regarding software development in the metering domain, S&D are of great importance to satisfy the requirements of the different catalogues and laws.

At this place the TERESA project may help to develop a trusted engineering process description, by providing unified S&D design patterns for model driven engineering techniques. The utilization of model driven engineering for software based embedded systems is a quite new approach in the metering sector. But this could lead to a trusted engineering process for meters, which would be validated instead of every new meter.

4.5 Further research approach in formalization of metering requirements

The first step of research work comprised the check for all relevant catalogues providing requirements for metering devices. The contents of the catalogues were summarized and all requirements related to security and dependability in the legally relevant part of metering software were extracted. The next step involves the check how far the requirements of the different catalogues overlap and to outline the commonalities. This also includes to resolve all dissents between the different requirements.

These condensed requirements form the basis for the further formalization approach. This approach will begin with a check for research related to this topic, that has already been carried out by other institutions. Depending on the results different formalization methods shall be adapted.

5 Discussion

In this section we summarize the discussions, positioning and comparison of existing works as presented in the previous sections. We then present a roadmap for the work to be carried out in Work package 7, from a scientific perspective as well as from the perspective of S&D engineering of embedded systems.

First, we review the main TERESA objectives:

- (1) to provide guidelines for the specification of sector specific RCES trusted computing engineering,
- (2) to define a trusted computing engineering approach that is suited to a four representative sectors: automotive, home control, industry control, and metering .

Conventionally "contemplative" and used as part of documentation, models tend to become "exploratory", and even "productive" if they are exploited by a process of development or simulation. Several initiatives have been conducted in this context, but are devoted to S&D engineering of embedded systems.

The main objective of model-driven engineering is to enforce traceability between different levels of abstractions and to ensure consistency between levels through verification tools to verify consistency or transformation tools to ensure consistency. TERESA will study specific challenges that may help improve gaps at this level for RCES trusted computing engineering:

- Variability support. Implementations derived from the same design pattern could differ in their details, while remaining similar in their principles.
- Code generation from a model, using transformation based on templates (patterns)
- Metrics integration to allow the association of assurance (certification/validation) levels with the various levels of abstraction
- Gathering and analysis of surveys carried out by other projects related to TERESA to characterize issues raised by TERESA contributions
- Generation of efficient test cases from patterns and fault models based on the model- driven approach and along with the traceability of requirements and test cases using SysML
- Extension of AADL for security modeling
- Extension of UML (MARTE) for security modeling.

These actions contribute to the definition of new systems engineering technologies with development centered on the models for trust RCES applications. From the model-driven point of view, the output of the project provides tool support for assisting the users to model their TERESA applications and to deploy and configure them on computing platforms. We focus on design tools with some view of the possible implementation frameworks providing run-time support, especially the dependability and security services and some additional control components.

Regarding S&D pattern integration and composition, we see the need for a method that conforms to many requirements. The methodology to be elaborated needs to allow for the same techniques to be applicable within various different domains, such as home control, industry control, metering and automotive, especially since the constraints on resources and environments may vary among them. In this way, each of the domains can benefit from advancements in the other domains and make use of inter-domain composition of patterns. Furthermore, the approach must be able to represent a complete set of S&D requirements needed in each of these domains. Also, the patterns described within the domains are not necessarily restricted to protocols, but may include organizational as well as platform related patterns. Finally, especially with regard to the possible diversity of S&D requirements, a suitable approach for composition should be independent of a specific attack models such as Dolev-Yao, and instead concentrate on security assurances provided by certain S&D patterns to agents acting within the system.

As we have seen in Section 3, many of the currently existing approaches address only a subset of S&D properties that are relevant for RCES (e.g. information flow properties). In addition, they require the definition of different attack models (e.g. universal composibility)which are restricted to a specific type of pattern composition (e.g. sequential composition as introduced by Andova et al.), or need to be extended for any new security mechanism that shall be handled (e.g. PCL). Hence it seems difficult to use any of these approaches for the work on integration and composition of S&D patterns, and it is not clear whether or how they can be extended in order to satisfy the above listed requirements.

In contrast, the Security Modeling Framework SeMF provides the necessary generality. With this framework, it is possible to specify all S&D requirements to be satisfied by a particular system based on one single formal model. The abstraction level of a formal model can be adapted to the specific requirements and is independent of any attack models. The first work on the composition of existing solutions has already been carried out and new work on the creation of generic composition processes has already been started.

Further research in the area of S&D pattern composition and integration will therefore target subjects such as:

- The description of S&D patterns detached from a concrete system as modules that can be substituted into “bigger” system definitions, i.e. during the integration of an S&D pattern into an overall system, or during the composition of two S&D patterns into a “bigger” S&D pattern,
- The possibilities to formalize the modules’ interface between S&D patterns and the integration target system,

- The description of properties provided by modules for every possible system integration, rather than properties of a simplified version of the system which neglect the environment of a certain S&D pattern,
- The specification of assumptions of an S&D pattern against the integration target system,
- The conditions, apart from strong typing, under which interference between several deployed S&D pattern can be disregarded,
- The derivation of sufficient conditions for the preservation of further S&D properties during abstraction, apart from the known conditions to support the approach of "validation through abstraction" of S&D pattern compositions,
- The possibility to adapt the current process for S&D pattern composition to the use of SeBB based verification instead of abstraction,
- The possibilities to utilize SeBB based verification techniques during the validation through abstraction of S&D patterns as well as their compositions,
- The provision of guidelines and best practices for security experts during the construction as well as validation of composed or integrated solutions.

The SeMF thereby enables not only the composition but also the verification of S&D properties and will be the basis for the work to be carried out in WP5, thus enabling a tight liaison between the two work packages.

In the *metering sector*, one important part of the way to bring a new device on the market is type approval. At present this is a very time consuming process, since most new devices have to be completely checked to verify if they fulfill all requirements. To help streamline this process, TERESA will study approaches to define an unified consistent design concept for the legally relevant part of the metering software.

Some important steps have been taken within this document. The requirement catalogues have been identified and summarized, and laws covering different parts of the metering domain have been taken into account as well. Requirements relevant to Security and Dependability aspects of metering software have been extracted. These requirements need to be transferred into a formal description. In future an automated validation of a development process for metering devices could take place based on these formalized requirements.

Further research and investigation work will deal with the following range of subjects:

- Investigate research that has already been carried out in the field of formalization of requirements
- Investigate forms of expression for formalized requirements so they can be used for the development of a trusted design process
- Investigate possibilities on how formalized requirements can be used for the subsequent validation of the trusted design process

- Investigate which parts of the trusted design process can be reliably validated and how validation can take place
- Choose a promising approach for the formalization of requirements
- Choose an appropriate language to describe/present the formalized requirements.

One possible candidate for the formalization of requirements is SeMF. Future work in WP7 will investigate its usefulness and applicability in the context of the metering domain.

In addition, there will be liaison with WP2, which comprises requirements from the application viewpoint, and with WP3, which deals with the engineering process viewpoint. Application domain requirements often lead to requirements towards the engineering process.

6 Bibliography

[1] R. Zurawski, Ed., *The Industrial Information Technology Handbook*. CRC Press, 2005.

[2] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: Design challenges," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 3, pp. 461–491, 2004.

[3] C. C. Jr, D. da Silva Jr, and A. Fernandes, "Hardware-software codesign of embedded systems," in *Integrated Circuit Design, 1998. Proceedings. XI Brazilian Symposium on*, 1998, pp. 2–8.

[4] J. Vidal, F. de Lamotte, G. Gogniat, P. Soulard, and J. Diguet, "A co-design approach for embedded system modeling and code generation with uml and marte," in *DATE*, 2009, pp. 226–231.

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

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

[7] F. Fondement and R. Silaghi, "Defining model driven engineering processes," in *in Proceedings of WISME*, 2004.

[8] D. Schmidt, "Model-driven engineering," *in IEEE computer*, vol. 39, no. 2, pp. 41–47, 2006.

[9] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, A.-W. . A. Press, Ed. Wesley, New York: Addison-Wesley / ACM Press, 2002.

[10] I. Crnkovic, "Component-based software engineering for embedded systems," in *Proceedings of the 27th international conference on Software engineering*. New York, NY, USA: ACM, 2005, pp. 712–713.

[11] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language*, ser. Center for Environmental Structure Series. New York, NY: Oxford University Press, 1977, vol. 2.

[12] G. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: a system of patterns*. John Wiley and Sons, 1996, vol. 1.

[13] J. Yoder and J. Barcalow, "Architectural patterns for enabling application security," in *Conference on Pattern Languages of Programs (PLoP 1997)*, 1998.

[14] M. Schumacher, *Security Engineering with Patterns - Origins, Theoretical Models, and New Applications*, ser. Lecture Notes in Computer Science. Springer, 2003, vol. 2754. [Online]. Available: http://www.springerlink.com/openurl.asp?genre=issue&issn=0302-9743&volume=2754

[15] F. Daniels, K. Kim, and M. A. Vouk, "The reliable hybrid pattern - a generalized software fault tolerant design pattern," in *Pattern Language of Programs Proceedings*, 2003.

[16] M. T. D. S. H. Giese, "Design of self-managing dependable systems with uml and fault tolerance patterns," in *WOSS '04: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*. New York, NY, USA: ACM, 2004, pp. 105–109.

[17] V. Giacomo, M. Felici, V. Meduri, D. Presenza, C. Riccucci, and A. Tedeschi, "Using security and dependability patterns for reaction processes," in *DEXA '08: Proceedings of the 2008 19th International Conference on Database and Expert Systems Application*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 315–319.

[18] N. Yoshioka, H. Washizaki, and K. Maruyama, "A survey of security patterns," *Progress in Informatics*, no. 5, pp. 35–47, 2008.

[19] A. W. Brown and K. C. Wallnau, "The current state of CBSE," *IEEE Software*, vol. 15, no. 5, pp. 37–46, 1998.

[20] I. Crnkovic, "Component-based software engineering—new challenges in software development," *Software Focus*, December 2001.

[21] N. Medvidovic and R. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Trans. Softw. Eng.*, vol. 26, no. 1, pp. 70–93, January 2000.

[22] M. Lehman, "Laws of software evolution revisited," in *EWSPT '96: Proceedings of the 5th European Workshop on Software Process Technology*. London, UK: Springer-Verlag, 1996, pp. 108–124.

[23] J. Magee and J. Kramer, "Dynamic structure in software architectures," in *SIGSOFT '96: Proc. of the 4th ACM SIGSOFT symposium on Foundations of software engineering*. New York, NY, USA: ACM Press, 1996, pp. 3–14.

[24] R. J. Allen, "A formal approach to software architecture," Ph.D. dissertation, Carnegie Mellon, School of Computer Science, January 1997, issued as CMU Technical Report CMU-CS-97-144.

[25] A. Özcan, S.Jean, and J.-B. Stefani, "Bringing ease and adaptability to mpsoc software design: A component-based approach," in *CASSIS*, 2005, pp. 118–137.

[26] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann, "Specification and analysis of system architecture using rapide," *IEEE Transactions on Software Engineering*, vol. 21, pp. 336–355, 1995.

[27] M. A. Abdallah, "Implementing the component-based software engineering in embedded systems," in *International SoC Design Conference*. Busan, South Korea: IEEE Computer Society, November 2008, pp. 399–402.

[28] K.-K. Lau and Z. Wang, "A taxonomy of software component models," in *Proc. 31st Euromicro Conference*. IEEE Computer Society Press, 2005, pp. 88–95.

[29] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, "The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems," *Softw. Pract. Exper.*, vol. 36, no. 11-12, pp. 1257–1284, 2006.

[30] T. Bures, P. Hnetynka, and F. Plásil, "Sofa 2.0: Balancing advanced features in a hierarchical component model." in *SERA*. IEEE Computer Society, 2006, pp. 40–48.

[31] F. Plásil and S. Visnovsky, "Behavior protocols for software components," *IEEE Trans. Softw. Eng.*, vol. 28, no. 11, pp. 1056–1076, 2002.

[32] F. Plásil, D. Balek, and R. Janecek, "SOFA/DCUP: Architecture for component trading and dynamic updating," in *Proc. of the Int. Conf. on Configurable Distributed Systems*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 43–52.

[33] J. Aldrich, C. Chambers, and D. Notkin, "Archjava: connecting software architecture to implementation," in *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*. New York, NY, USA: ACM, 2002, pp. 187–197.

[34] F. Hacklinger, "Java/a - taking components into java," in *IASSE*, 2004, pp. 163–168.

[35] B. Hamid, A. Radermacher, A. Lanusse, C. Jouvray, S. Gerard, and F. Terrier, "Designing fault-tolerant component based applications with a model driven approach." in *SEUS*, ser. Lecture Notes in Computer Science, U. Brinkschulte, T. Givargis, and S. Russo, Eds., vol. 5287. Springer, 2008, pp. 9–20.

[36] OMG, "A uml profile for marte: Modeling and analysis of real-time embedded systems,beta 2," http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf, June 2008.

[37] D. Serrano, A. Mana, and A.-D. Sotirious, "Towards precise and certified security patterns," in *Proceedings of 2nd International Workshop on Secure systems methodologies using patterns (Spattern 2008)*. Turin, Italy: IEEE Computer Society, September 2008, pp. 287–291.

[38] S. Fritsch and S. Clarke, "Timeadapt: timely execution of dynamic software reconfigurations," in *Proceedings of the 5th Middleware doctoral symposium*. New York, NY, USA: ACM, 2008, pp. 13–18.

[39] A. Rasche and A. Polze, "Dynamic reconfiguration of component-based real-time software," in *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 347–354.

[40] H. Lu, W. Tang, and X. Wang, "A dynamic customizable model for component-based embedded systems," in *International Conference on Audio, Language and Image Processing*. IEEE Computer Society, July 2008, pp. 220–224.

[41] Y. Guo, K. Sierszecki, and C. Angelov, "A (re)configuration mechanism for resource-constrained embedded systems," in *Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 1315–1320.

[42] I. Crnkovic, "Component-based approach for embedded systems," in *Proceedings of 9th International Workshop on Component-Oriented Programming*, 2004.

[43] (2006) Corba component model specification, v4.0. http://www.omg.org/spec/CCM.

[44] (2000) Simple object access protocol (soap), v1.1. http://java.sun.com/products/archive/jaas/.

[45] (2008) Autosar gbr. autosar technical overview, v2.2.2, r3.1. http://www.autosar.org.

[46] J. Fleischmann, K. Buchenrieder, and R. Kress, "Codesign of embedded systems based on java and reconfigurable hardware components," *Design, Automation and Test in Europe Conference and Exhibition*, p. 768, 1999.

[47] *IEEE Standard SystemC Language Reference Manual*, IEEE Computer Society, 2005.

[48] "http://www.systemc.org."

[49] Y. Wang, X. Zhou, B. Zhou, L. Liang, and C.Peng, "A mda based soc modeling approach using uml and systemc," in *CIT '06: Proceedings of the Sixth IEEE International Conference on Computer and Information Technology*, 2006, p. 245.

[50] SAE, "Architecture analysis & design language (aadl)," http://www.sae.org/technical/standards/AS5506A, January 2009.

[51] J. Perez, M. Azkarate-askasua, and A. Perez, "Codesign and simulated fault injection of safety-critical embedded systems using systemc (to appear)," 2010.

[52] (2007) Omg unified modeling language (omg uml), infrastructure, v2.1.2. omg document number: formal/2007-11-04. .

[53] S. Demathieu, F. Thomas, C. André, S. Gérard, and F. Terrier, "First experiments using the uml profile for marte," in *Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*. Washington, DC, USA: IEEE Computer Society, May 2008, pp. 50–57.

[54] S. Gérard and B. Selic, "The uml - marte standardized profile," 2008.

[55] F. Thomas, S. Gérard, J. Delatour, and F. Terrier, "Software real-time resource modeling," in *FDL*, 2007, pp. 231–236.

[56] S. Taha, A. Radermacher, S. Gerard, and J.-L. Dekeyser, "An open framework for detailed hardware modeling," in *SIES*, 2007, pp. 118–125.

[57] A. Koudri, D. Aulagnier, D.Vojtisek, P. Soulard, C. Moy, J. Champeau, J. Vidal, and J.-C. L. Lann, "Using marte in a co-design methodology," in *Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile workshop co-located with DATE'08*, 2008.

[58] OMG, "Omg systems modeling language (omg sysml)," http://www.omg.org/spec/SysML/1.1/, November 2008.

[59] H. C. K. Forsberg, H. Mooz, "Visualizing project management, a model for business and technical success." Wiley; 2 edition, 2000.

[60] P. Koopman, "Embedded system design issues (the rest of the story)," *Computer Design, International Conference on*, vol. 0, p. 310, 1996.

[61] R. Zurawski, "Embedded systems in industrial applications - challenges and trends," in *SIES*, 2007.

[62] H. Espinoza, D. Cancila, B. Selic, and S. Gérard, "Challenges in combining sysml and marte for model-based design of embedded systems," in *ECMDA-FA '09: Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications*. Springer-Verlag, 2009, pp. 98–113.

[63] H. Kopetz, "The complexity challenge in embedded system design," in *ISORC*, 2008, pp. 3–12.

[64] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and M. G. Harbour, "Influence of different system abstractions on the performance analysis of distributed real-time systems," in *EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software*. New York, NY, USA: ACM, 2007, pp. 193–202.

[65] A. Youssef, Y. Crouzet, A. de Bonneval, J. Arlat, J.-J. Aubert, and P. Brot, "Communication integrity in networks for critical control systems," in *EDCC*, 2006, pp. 23–34.

[66] R. Kamal, "Embedded systems: Architechture, programming and design." McGraw-Hill Science/Engineering/Math, 2006.

[67] J. O. Kephart, "Research challenges of autonomic computing," in *ICSE*, 2005, pp. 15–22.

[68] A. G. Ganek, C. P. Hilkner, J. W. Sweitzer, B. A. Miller, and J. L. Hellerstein, "The response to it complexity: Autonomic computing," in *NCA*, 2004, pp. 151–157.

[69] V. Garg, *Elements of distributed computing*. John wiley & sons. New york, 2002.

[70] H. Attiya and J. Welch, *Distributed computing, fundamentals, simulations and advanced topics*. Edition McGraw-hill international (UK) limited, 1998.

[71] T. Henzinger, "Two challenges in embedded systems design: Predictability and robustness," *Philosophical Transactions of the Royal Society A*, vol. 366, pp. 3727–3736, 2008.

[72] F. Parain, "Region-based memory management for real-time java," in *ISORC '01: Proceedings of the Fourth International Symposium on Object-Oriented Real-Time Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2001, p. 387.

[73] H. Mantel, "On the composition of secure systems," in *IEEE Symposium on Security and Privacy*. Citeseer, 2002, pp. 88–104.

[74] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, 2000.

[75] ——, "Universally composable security: A new paradigm for cryptographic protocols," in *focs*. Published by the IEEE Computer Society, 2001, p. 136.

[76] M. Bellare, R. Canetti, and H. Krawczyk, "A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract)," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 419–428.

[77] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic, "Abstraction and refinement in protocol derivation," in *Proceedings of 17th IEEE Computer Security Foundations Workshop*. Citeseer, 2004, pp. 30–45.

[78] C. Cremers, "On the protocol composition logic pcl," in *ASIACCS '08: Proceedings of the 2008 ACM symposium on Information, computer and communications security*. New York, NY, USA: ACM, 2008, pp. 66–76.

[79] F. Thayer, J. Herzog, and J. Guttman, "Mixed strand spaces," in *Proceedings of the 12th IEEE workshop on Computer Security Foundations*. IEEE Computer Society, 1999, pp. 72–82.

[80] S. Andova, C. Cremers, K. Gjøsteen, S. Mauw, S. Mjølsnes, and S. Radomirovic, "A framework for compositional verification of security protocols," *Information and Computation*, vol. 206, no. 2-4, pp. 425–459, 2008.

[81] S. Mauw and C. Cremers, "Operational Semantics of Security Protocols," *Lecture notes in computer science*, vol. 3466, pp. 66–89, 2005.

[82] J. Guttman, "Cryptographic Protocol Composition via the Authentication Tests," in *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*. Springer, 2009, pp. 303–317.

[83] H. Choi, "Security protocol design by composition," *Cambridge University, UK, Technical report*, vol. 657, pp. 1476–2986, 2006.

[84] L. G., "A hierarchy of authentication specifications," in *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW'97)*. IEEE Society Press, 1997, pp. 31–43.

[85] C. Rudolph, "A model for secure protocols and its application to systematic design of cryptographic protocols," Ph.D. dissertation, Queensland University of Technology, 2001.

[86] S. Mödersheim and L. Viganò, "Secure pseudonymous channels," *Computer Security – ESORICS 2009*, pp. 337–354, 2009.

[87] S. Mödersheim, "Algebraic properties in Alice and Bob notation," in *2009 International Conference on Availability, Reliability and Security*. IEEE, 2009, pp. 433–440.

[88] A. Fuchs, S. Gürgens, and C. Rudolph, "Towards a Generic Process for Security Pattern Integration," in *Trust, Privacy and Security in Digital Business – 6th International Conference, TrustBus 2009, Linz, Austria, September 3–4, 2009, Proceedings*. Springer, 2009.

[89] "Serenity, system engineering for security & dependability," www.serenity-project.org, 2006.

[90] A. Fuchs, S. Gürgens, and C. Rudolph, "On the Security Validation of Integrated Security Solutions," in *Emerging Challenges for Security, Privacy and Trust: 24th Ifip Tc 11 International Information Security Conference, SEC 2009, Pafos, Cyprus, May 18-20, 2009, Proceedings*. Springer, 2009, pp. 190–201.

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

[92] S. Gürgens, P. Ochsenschläger, and C. Rudolph, "Abstractions preserving parameter confidentiality," in *Computer Security – ESORICS 2005*. Springer, 2005, pp. 418–437.

[93] European Parliament, "Directive 2006/32/ec of the european parliament and of the council of 5 april 2006 on energy end-use efficiency and energy services and repealing council directive 93/76/eec," http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2006:114:0064:0064:EN:PDF, 4 2006.

[94] ——, "Directive 2004/22/ec of the european parliament and of the council of 31 march 2004 on measuring instruments," http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2004:135:0001:0080:EN:PDF, 3 2004.

[95] European Council, "Council directive 76/891/eec of 4 november 1976 on the approximation of the laws of the member states relating to electrical energy meters," 11 1976.

[96] WELMEC, "Welmec 7.2 issue 4 software guide," http://www.welmec.org/fileadmin/user_files/publications/7-2_Issue4.pdf, 5 2009.

[97] WELMEC, "Welmec geography," http://www.welmec.org/welmec/welmec-tour/welmec-geography.html, 12 2009.

[98] P.-T. B. Physical-Technical Federal Institute of Germany (PTB), "Requirements for software-based metering devices," http://www.ptb.de.

[99] OIML, "Oiml d 31 general requirements for software controlled measuring instruments," http://www.oiml.org/publications/D/D031-e08.pdf, 2008.

[100] Bundesministerium der Justiz, "Signaturgesetz - sigg," http://www.gesetze-im-internet.de/bundesrecht/sigg_2001/gesamt.pdf, 5 2001.

[101] ——, "Gesetz über das meß- und eichwesen (eichgesetz)," http://www.gesetze-im-internet.de/bundesrecht/eichg/gesamt.pdf, 7 1969.

[102] ——, "Bundesdatenschutzgesetz - bdsg," http://www.gesetze-im-internet.de/bundesrecht/bdsg_1990/gesamt.pdf, 12 1990.

[103] ——, "Telekommunikationsgesetz (tkg)," http://bundesrecht.juris.de/bundesrecht/tkg_2004/gesamt.pd 6 2004.

[104] ——, "Gesetz über den datenschutz bei telediensten (teledienstedatenschutzgesetz - tddsg)," 7 1999.

[105] "Teresa project," http://www.teresa-project.org, 4 2010.

[106] C. Ruland, "Secure online metering for a liberalized energy market secure egovernment web services," in *Proceedings of the 27th international conference on Software engineering*. Hershey, PA, USA: Idea Group Publication, 2006, pp. 97–110.

[107] OMG, *MDA Guide*, 1st ed., 2003.

[108] M. W. Christoph Ruland, "Das selma-projekt, konzepte, modelle, verfahren." PTB-IT-12, Physikalisch-Technische Bundesanstalt Braunschweig und Berlin, Norbert Zisky, pp. 117–164.

[109] N. Z. Christoph Ruland, Luigi Lo Iacono, "Secure transfer of measurement data in open systems computer standards & interfaces," pp. 311–326.

[110] SYM2-project, "Functional specification document for electricity metering devices," http://www.sym2.org.

[111] B. Schneier, *Applied Cryptography*, 2nd ed. John Wiley & Sons, 1996.

[112] S. H. Reinhard Höhn, *Das V-Modell XT. Grundlagen, Methodik und Anwendungen*. Springer Verlag.

[113] Bundesministerium der Justiz, "Gesetz über die elektrizitäts- und gasversorgung (energiewirtschaftsgesetz - enwg)," http://www.gesetze-im-internet.de/bundesrecht/enwg_2005/gesamt.pdf, 7 2005.