Master's Thesis

Optimization of the Security-Related Development Process for Secure Smart Cards

Philip Baumgartner, BSc

Institute for Technical Informatics
Graz University of Technology

in cooperation with NXP Semiconductors Austria GmbH

Assessor: Dipl.-Ing. Dr.techn. Christian Kreiner
Supervisor: Dipl.-Ing. Wolfgang Raschke

Graz, September 2014

Masterarbeit

Optimierung des sicherheitsrelevanten Entwicklungsprozesses für sichere Smartcards

Philip Baumgartner, BSc

Institut für Technische Informatik
Technische Universität Graz

in Kooperation mit NXP Semiconductors Austria GmbH

Begutachter: Dipl.-Ing. Dr.techn. Christian Kreiner
Betreuer: Dipl.-Ing. Wolfgang Raschke

Graz, September 2014

Abstract

The development process of integrated circuits for secure smart card solutions basically consists of two processes that need to be handled in parallel: the product development process on the one hand and the security evaluation process on the other. Both processes are essential parts of the product development. The second one is especially needed for the Common Criteria for Information Technology Security Evaluation, and is absolutely necessary for getting a product certified. The structure of these two processes influences the developers performance and the effort needed for documentation. In order to maximize efficiency it is absolutely necessary to combine these two processes and model a clean structured and well elaborated unified process that covers all aspects of development and security evaluation.

This master's thesis covers the design of the optimized development process with a special focus on agile development and automatic generation of reusable documentation parts. The implementation part contains a prototype of a requirements and specification editor based on an XML model and a mechanism for synchronizing models from several sources.

This project ran in cooperation with NXP Semiconductors Austria GmbH.

Kurzfassung

Der Entwicklungsprozess von integrierten Schaltkreisen für sichere Smartcard Lösungen besteht im Wesentlichen aus zwei Prozessen, die parallel durchgeführt werden müssen: der Entwicklungsprozess auf der einen Seite und der Sicherheitsevaluierungsprozess auf der anderen. Beide Prozesse sind essentielle Teile bei der Produktentwicklung. Speziell der zweite Prozess wird für die Zertifizierung der Produkte nach dem Common Criteria Standard benötigt und ist unbedingt notwendig um ein Produkt zertifizieren zu können. Die Struktur dieser beiden Prozesse hat dabei einen essentiellen Einfluss auf die Zeit, die Entwickler in Entwicklung und Dokumentation investieren müssen. Um möglichst effizient arbeiten zu können, ist es notwendig diese beiden Prozesse zu kombinieren und in einem sauber definierten und gut durchdachten einheitlichen Prozess abzubilden, der schließlich alle Aspekte der Entwicklung und Sicherheitsevaluierung abdeckt.

Diese Masterarbeit behandelt das Design des optimierten Entwicklungsprozesses, wobei besonderes Augenmerk auf die agile Entwicklung und die automatische Generierung wiederbenutzbarer Dokumentationsteile gelegt wird. Die Implementierung beinhaltet einen Prototypen eines Anforderungs- und Spezifikationseditors basierend auf einem XML Modell und einen Mechanismus zur Synchronisation von Modelldaten aus unterschiedlichen Quellen.

Dieses Projekt wurde in Kooperation mit NXP Semiconductors Austria GmbH durchgeführt.

Danksagung

An dieser Stelle möchte ich mich bei allen Bedanken, die es mir ermöglicht haben heute an dieser Stelle zu stehen und mich während des gesamten Studiums immer tatkräftig unterstützt haben. Mein besonderer Dank geht dabei an meine Eltern, die es mir ermöglicht haben diesen Weg zu gehen und in dieser Zeit immer hinter mir standen. Außerdem möchte ich mich bei all meinen Freunden bedanken, besonders bei Andi und Mario, mit denen ich gemeinsam die letzten sieben Jahre gemeistert habe und die mittlerweile auch zu guten Freunden geworden sind. Mein weiterer Dank gilt Christian Kreiner und Wolfgang Rasche, die mich bei der Durchführung dieser Arbeit immer wieder mit konstruktiven Gedanken tatkräftig unterstützt haben und mir ermöglichten an einem interessanten Thema zu arbeiten. Außerdem gilt mein Dank natürlich auch Martin Schaffer, der die Kooperation mit NXP möglich gemacht hat und Georg Stütz, sowie Rainer Doppelreiter für die Unterstützung seitens NXP und auch die ein oder andere unterhaltsame Stunde im Büro während der Durchführung dieser Arbeit.

Graz, September 2014 Philip Baumgartner

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen / Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, 04.09.2014 .....

Signature

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 General Introduction

Secure smart cards have been invented in the beginning of the 1970's for the purpose of storing confidential data on an identification medium. Nowadays, smart cards are used for a huge number of applications such as for electronic passports [1], ticketing, animal identification, access control cards or contactless payment systems [2]. According to the World Payments Report 2013 [3] already in 2012 about 333 billion cashless payments have been processed.

The development process of such Integrated Circuits (IC) for secure smart cards is a complex process that basically consists of two parts: the development process and the security evaluation process [4]. Since the products are used for security relevant applications, it is necessary to certify the final product. In this evaluation process several institutions are involved: the developing company with its developers, the evaluation facility and the certification authority of the country, which issues the certificate. The task of the evaluation facility is to investigate and evaluate the implementation of the product. For this security evaluation there exists several standards. The so-called Common Criteria for Information Technology Security Evaluation standard (CC) [5] defines criteria that must be fulfilled by the product to pass the certification process. Chapter 3 gives an overview of this standard. A product satisfying these criteria gets a certificate that ensures the product to be secure. Especially for the security evaluation performed by the evaluation facility evidence must be provided for the security of the product. Therefore, as a developer you have to make sure your design is well structured. In addition you have to prove that it fulfills the security requirements and its resistance against attacks.

This master's thesis strives to model a process and to find a suitable tooling environment for the development of secure smart cards. This includes the support of the requirements and specification management, of the implementation and of the documentation and certification process.

This work is structured as follows. Chapter 1 describes the starting point, and the stakeholder analysis that has been performed to get familiar with the current development situation and to

identify possible problems. Chapter 2 provides a summary of these problems. Afterwards, existing technologies for producing engineering documents and design documentation have been analyzed and evaluated. The results of this related work analysis are described in Chapter 3. Based on the problem definition, goals and requirements for the optimized development process have been derived, which are listed in Chapter 4. This chapter also discusses possible solutions and gives an overview of the entire concept and design. Chapter 5 provides a description of the prototype implementation of a requirements management system. This implementation has been evaluated and the results are described Chapter 6. Finally, Chapter 7 summarizes the work, describes the known limitations and provides information about ongoing work.

1.2 Starting Point

This master's thesis has been created in cooperation with NXP Semiconductors Austria GmbH (NXP). This section describes the starting point for this work by analyzing the current situation of the development and security evaluation process in this company.

In the course of the last years the complexity of the developed ICs increased enormously. In order to deal with this complexity existing parts of previous developments need to be reused. Additionally the trend is towards the development of product lines [6] that consist of a number of product derivates. Product derivates are products that have similar structure and features, but differ in some detail. As stated in [7], product lines help minimizing the effort among others in doing the requirements engineering, creating the architecture and design as well as developing the test plan and documentation of the whole project and implementation. Furthermore, they see product lines as a strategic reuse of existing parts of products. Such an approach makes it necessary to change the previous way of work and to optimize the development process. Simple text processing software such as Microsoft Word cannot be longer used to document the design and security evaluation of products. Because of the high effort for maintenance, inconsistencies in the documentation cannot be detected easily.

Especially for security relevant products it is necessary to have a well structured design and a consistent documentation.

Due to all these circumstances, the development process needed to be optimized. Thus, in the first step a stakeholder analysis has been performed, which aims to find out the stakeholders involved in the product development process and their roles and impact on the project. This facilitates identifying the problems and defining the according actions to be taken.

1.3 Stakeholder Analysis

This section describes the stakeholder analysis [8] of the currently used requirements management system (RMS) IBM Rational DOORS (DOORS)[1], which is used to store requirements and specification artifacts. Each stakeholder has different requirements for the development process.

During the stakeholder analysis the stakeholders shown in Figure 1 have been identified and are described below.

- **Program Manager, Project Manager**
  Monitors the project progress and specification changed within a certain period of time.
- **Configuration Manager**
  Defines project templates for a dedicated product.
- **System Architect**
  Defines project requirements on different abstraction levels.
- **Test Architect**
  Defines test specification.
- **Security Architect / Common Criteria Engineer**
  Uses DOORS for storing information necessary for the security evaluation process.
- **Developer**
  Defines interface specification for hardware and firmware; documents the implementation.
- **DOORS Trainer**
  Informs users about the correct usage of the requirements management tool.
- **DOORS DB Administrator**
  Administrates and maintains the DOORS database.
- **Documentation Environment**
  A system interacting with DOORS in order to generate parts of the documentation automatically.

[1] http://www.ibm.com/software/products/de/ratidoor

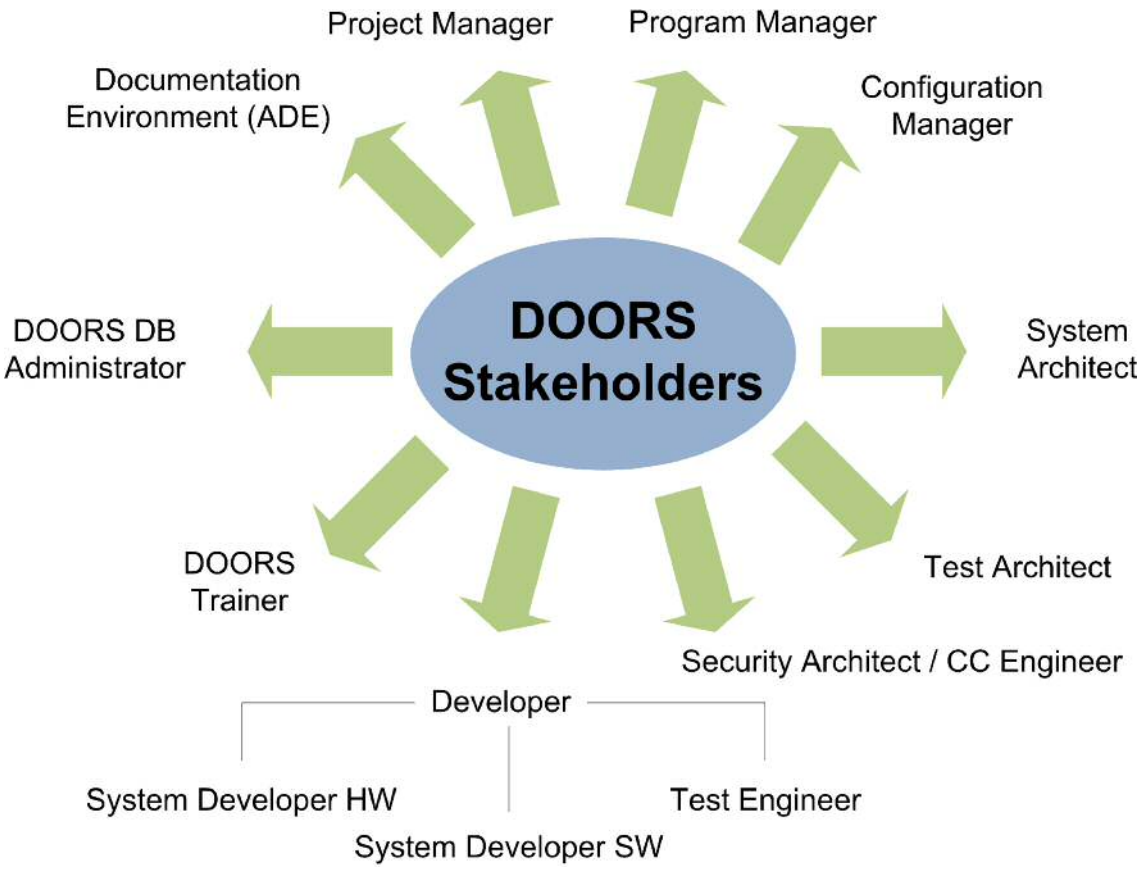

Figure 1: Stakeholder analysis of the RMS DOORS.

After the identification of the groups, roles and systems involved in the development process, a questionnaire has been prepared that is listed below. The aim is to evaluate the problems of the current development process together with the project members. Since most of the problems are related to the currently used requirements management system (RMS), the questions put a focus on this field.

1. What is your role in the development process?
2. For which purpose do you use DOORS and how?
3. What bothers you most when you are working with DOORS?
4. Did you identify any missing functionality in DOORS and which one?
5. How is your preferred way of working when using a requirements management system?

The result of this evaluation shows that most of the problems are related to the currently used RMS. The identified problems are discussed in Chapter 2.

Chapter 2

Problem Definition

This chapter provides a short overview of the main problems of the development process that is going to be optimized. The problems have been identified by evaluating the results and discussions of the questionnaire described in Section 1.3. Possible solutions are proposed in Chapter 4 of this work.

2.1 Development Process and Security Evaluation

[Problem 1]: Separated processes for development and security evaluation.

One of the main problems is the existence of two separated processes for development and security evaluation. These processes basically depend on each other, but in the current situation they are completely disconnected. Thus, there are no links between artifacts of the security evaluation process and the according artifacts in the development process. This makes it difficult to analyze dependencies between artifacts of the two processes. However, such analyses are necessary for the security evaluation and hence, for the certification of a product.

These two separated processes are shown in Figure 2. The left part of the figure shows the security evaluation process, whereas the development process is depicted on the right hand side.

The development process depicts the entire process from the high-level requirements for a certain product on a high level of abstraction and a customer's point of view down to the definition of all necessary scopes and subscopes needed for the implementation of the product. This includes all interface specifications, test specifications and user guidance on each level of abstraction. Since the CC terminology is used in these descriptions, the word scope is used here to describe a single hardware or software module implementing a certain set of features.

The documents are created as part of the security evaluation process and provide an evidence for the security of the developed product.

The structure of this process is defined by the Common Criteria for Information Technology Security Evaluation standard [5], which specifies criteria for the evaluation of secure IT systems and is defined in the DIN ISO/IEC 15408-1...3. Section 3.1 gives a short overview of this standard and the according process.

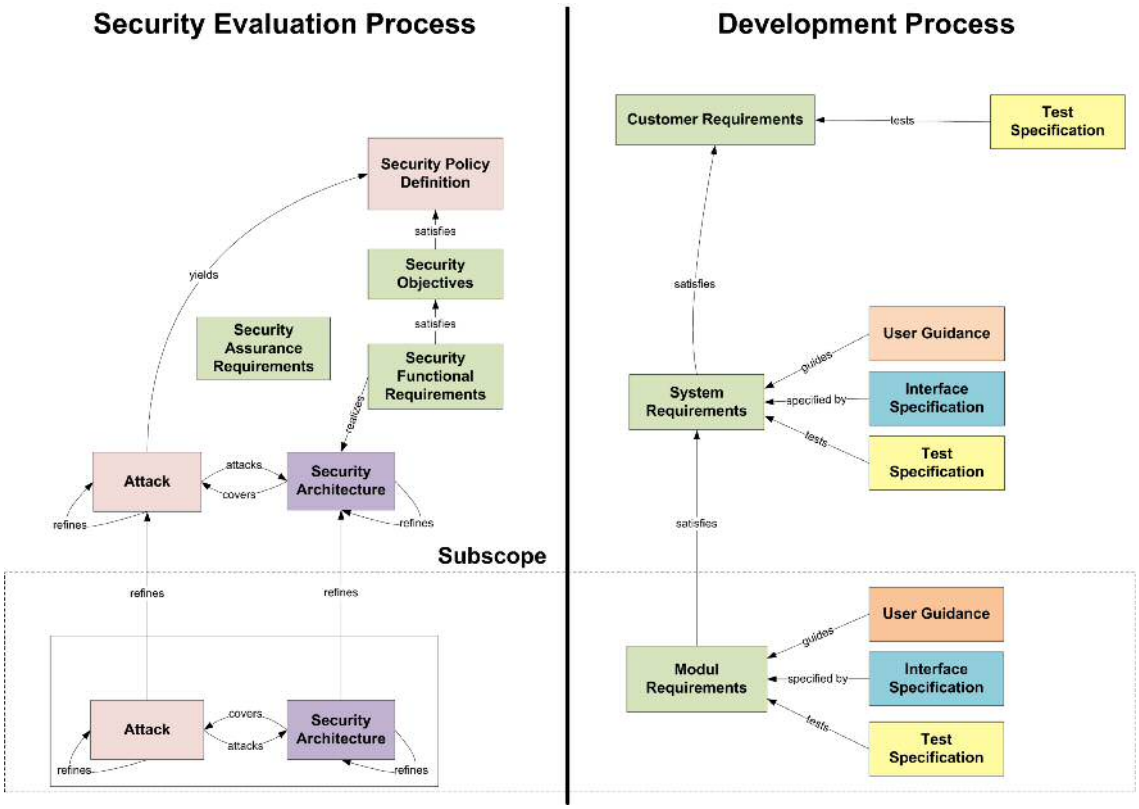

Figure 2: Overview of the two separated processes for development and security evaluation.

2.2 Reuse of Requirements and Specification

[Problem 2]: Reuse of requirements and specification in the documentation is not possible.

Another problem is the missing functionality for referencing requirements and specification artifacts, defined in the RMS, in the documentation. For the design documentation and the security evaluation several requirements and specification artifacts need to be referenced or listed at different locations in the documentation. As a result, equal text parts need to be copied and pasted on different locations in the documentation.

The following example should make the scenario more clear. Assume you have a firmware header file with interfaces defined in it. An interface consists of the interface method, its parameters and a return value. In the design documentation it is necessary to describe the interfaces and all its attributes. For this purpose it is necessary to copy the interface from the source code to the design documentation, where its description is going to be placed. Furthermore, it might also be required to refer to the interface at some other point in the documentation. Thus, again you have to copy the interface definition to the location in the document, where it is needed again.

This way of work is a real copy and paste nightmare. Redundancy and inconsistency are the consequences. Furthermore, product maintenance and product changes become a very time-consuming and error-prone process.

2.3 IBM Rational DOORS as Requirements Management System

In the current development process the commercial software IBM Rational DOORS (DOORS) is used for requirements and specification management. This software has some major disadvantages when it is used for the development of secure smart cards. The main problems are discussed in the following.

DOORS organizes data in modules that basically consist of a simple data table. An arbitrary number of columns can be configured. A single column represents a data type that can be defined by the user. Additionally, a level can be assigned to each row, which makes it possible to store hierarchical information.

[Problem 3]: Missing data validation.

One of the problems is the missing data validation. Sometimes it is necessary to define a set of attributes for a data object, depicted as row in DOORS, where information needs to be filled in by the user mandatorily. However this is a behavior that cannot be enforced by this software. Instead, all columns can be seen as optional and thus they can also be kept empty. It is also not possible to define different types of data objects with different attributes. Hence, an attribute is always represented as column, and each column is valid for each row. Thus, it is not possible to define different data objects (rows) with different attributes, instead all available attributes can be used for all data objects.

[Problem 4]: Low performance of data export.

Another disadvantage of DOORS is the low performance for data export. Exporting DOORS modules is a very time-consuming process and needs to be performed in order to convert module data of the proprietary DOORS database to another file format.

[Problem 5]: High round trip time for firmware developers.

Interface specification and other specification artifacts are stored in DOORS. Especially for the software interfaces the C header files are generated by an automated process that works as shown in Figure 3. In the first step a DOORS export process is triggered and the interface specification is exported to a CSV file. Then, a conversion process is applied that converts the resulting CSV file into a defined XML format. This XML file is then used as input for the header file generation engine, which produces the final C header file for the developer. As a result, each time a developer wants to change an interface the specification must be changed in the according module in DOORS and the header file generation process needs to be started again. Since the DOORS export process is a very time-consuming process the round trip time for the software development increases tremendously. Especially for the agile firmware development this is an enormous problem.

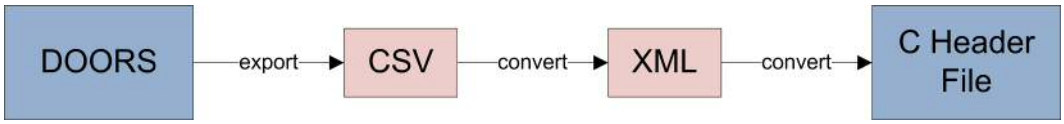

Figure 3: Simplified header file generation process.

[Problem 6]: Proprietary database and DXL.

DOORS is based on a proprietary database and is thus very inflexible. For creating queries a user must be familiar with the complex query language DXL, which is proprietary as well. Thus it needs much time to create suitable queries and extensions for DOORS.

[Problem 7]: Missing teamwork opportunity.

DOORS produces tremendous amounts of data when more than one user work on the same module concurrently. To avoid this, it is necessary to lock a module before modifying its data. Consequently only one user can work on one module at the same time, which slows down the development process. This is a very inefficient way of work and unsuitable for agile development.

[Problem 8]: Missing versioning and branching functionality.

Versioning and branching are not supported. DOORS stores only the history of single data records in a module, but there is no simple way to backup or store the current version of a module separately. This would be necessary for instance for the recertification of products that are already brought on the market. Branching is necessary for the development of product

families. A product family contains product derivates that have a similar structure and features but differ in some details. This cannot be represented in DOORS and turns out to be another disadvantage. Therefore, an efficient reuse of existing requirements and specification in a new product is also not possible in an appropriate way.

2.4 Summary of the Problems

Table 1 summarizes the problems that have been identified in the development process. Possible solutions for these problems are provided in Chapter 4.

Problem	Description
Problem 1	Separated processes for development and security evaluation
Problem 2	Reuse of requirements and specification in the documentation is not possible
Problem 3	DOORS: Missing data validation
Problem 4	DOORS: Low performance of data export
Problem 5	DOORS: High round trip time for firmware developers when using DOORS as *single-point-of-source* for interface specification
Problem 6	DOORS: Proprietary database and complex query language DXL
Problem 7	DOORS: Missing teamwork opportunity
Problem 8	DOORS: Missing versioning and branching functionality

Table 1: Summary of the identified problems of the development process.

Chapter 3

Related Work

This chapter presents the results of the related work analysis. It gives a short overview of the CC standard, compares existing frameworks and technologies for producing engineering documents and discusses a number of differencing algorithms regarding their suitability for calculating comparison models of design artifacts.

3.1 Common Criteria - a Short Overview

The Common Criteria for Information Technology Security Evaluation is an international standard defining criteria for the evaluation and certification of secure IT systems. Herrmann [9] describes the goals of this project as follows:

"The goal of the CC project was to develop a standardized methodology for specifying, designing, and evaluating IT products that perform security functions which would be widely recognized and yield consistent, repeatable results. In other words, the goal was to develop a full-lifecycle, consensus-based security engineering standard."

The CC standard is basically divided into three parts [5]:

- **Part 1:** Introduction and General Model.
- **Part 2:** Security Functional Requirements.
- **Part 3:** Security Assurance Requirements.

Part 1 describes terminology and concepts, contains a description of the CC methodology, the history of development and CC sponsoring organizations. Part 2 contains a catalog of standardized Security Functional Requirements (SFRs) and part 3 lists standardized Security Assurance Requirements (SARs).

CC [9] divides the security specification into two parts, which are described below.

Protection Profile (PP). This is an implementation independent definition of security functional requirements of the TOE (Target of Evaluation), which is any physical implementation such as firmware, software, hardware etc. that is going to be evaluated. It can be seen as a generic security target for a certain class of products. The protection profile for secure smart cards [10] has been developed by Eurosmart in cooperation with Inside Secure, Infineon Technologies AG, NXP Semiconductors Germany GmbH and STMicroelectronics.

Security Target (ST). This is an implementation dependent design that describes security mechanisms, features and functions to fulfill the requirements defined in the PP for a certain TOE.

As stated in the CC Standard [5] *"[...] the security problem definition defines the security problem that is to be addressed"*. This includes the threats and assumptions about the TOE and its operational environment.

The security objectives are described as *"[...] a concise and abstract statement of the intended solution to the problem defined by the security problem definition"*.

"The SFRs are a translation of the security objectives for the TOE. They are usually at a more detailed level of abstraction, but they have to be a complete translation (the security objectives must be completely addressed). The CC requires this translation into a standardized language [...]".

"The SARs are a description of how the TOE is to be evaluated. This description uses a standardized language [...]".

A certification of a product can be performed on different levels, so-called evaluation assurance levels (EAL). Table 2 gives an overview of all EALs defined in the CC standard. The precision of investigation of the developed product and the depth of documentation needed for the certification increases according to the EAL.

When a product is going to be certified, it is submitted to an independent evaluation laboratory, which evaluates the developed product by applying different attacks. All these attack paths are then rated in two different manners:

- Effort and knowledge needed to identify an attack.
- Effort and knowledge needed to exploit this known attack to attack all instances of this device.

EAL	Description
EAL1	Functionally tested
EAL2	Structurally tested
EAL3	Methodically tested and checked
EAL4	Methodically designed, tested and reviewed
EAL5	Semiformally designed and tested
EAL6	Semiformally verified design and tested
EAL7	Formally verified design and tested

Table 2: Overview of CC evaluation assurance levels [5].

The more difficult it is to perform a specific attack, the higher is the number of points it is rated with.

For this purpose the evaluation laboratory must be provided with IC samples and the according documentation, which contains a detailed description of the design and the results of the security evaluation process.

Not only the product itself is rated, but also the development process. This includes the development site and its offices, the IT infrastructure and development tools and the production process. This means that knowledge facilitating the attack needs to be kept confidential to get additional points for certain attack paths.

The goal is to reach a maximum amount of points or at least the minimum required amount to get the product certified.

After finishing the evaluation, the results are reported to the certification authority, which then issues the according certificate.

3.2 Different Approaches for Model Differencing

Model differencing or in other words the calculation of comparison models, which shows the differences of two data models, is not a trivial task and still a quite young field of research. *"Model differencing involves a number of steps starting with identifying matching model elements, calculating and representing their differences, and finally visualizing them in an appropriate way"* [11].

Related to requirements management there are several use cases that have been identified, which require the calculation of comparison models. Some of them are described in Chapter 4. Most of the conventional differencing engines, such as the UNIX differencing tool diff[2], compare files on a line by line basis. For text files such as documentation or source code this is a proper way to find differences and the methods and algorithms for this purpose are well elaborated, but for data models this is not an applicable way of work. Instead much more sophisticated algorithms are needed that are aware of the meta-model or at least the language the model is described in. The meta-model describes the structure how data is stored within the model.

As Kolovos et al. [11] stated in their paper, the process of model differencing can be divided into three basic tasks:

- **Calculation**: Calculation of the model differences and creation of the comparison model.
- **Representation**: Storing the result of the calculation for further use.
- **Visualization**: Process data in a way to get a human-readable notation.

The calculation task can again be divided into two phases:

- **Matching phase**: Finding correspondences between the models.
- **Differencing phase**: Finding differences between two versions of the same element.

They also describe several basic approaches for model matching, which are summarized below.

Static Identity-Based Matching. The static identity-based matching approach uses unique identifiers for the identification of corresponding elements. This has the advantage of being very fast and requires no user configuration. The disadvantage of this approach is the missing opportunity to compare models that have been developed independently from each other.

Signature-Based Matching. Another approach is the so called signature-based matching. A implementation following this approach calculates a signature for each model element and compares them. Thus, instead of having a unique identifier, a signature calculation function must be provided by the user.

[2] http://www.gnu.org/software/diffutils

Similarity-Based Matching. While both of the approaches explained above return a hard decision if two elements match to each other, the next approach measures only the similarity of two elements and creates the matching out of these results. Since not all features of an element have the same importance for the calculation of the matching, this similarity-based approach uses weighted features to calculate the comparison of elements.

Custom Language-Specific Matching. Finally, they explained custom language-specific matching algorithms, where the user must specify the entire matching algorithm, which is not a trivial task. This approach has the advantage of being able to make use of the semantics of the target language and thus to get more precise results.

What all the presented approaches have in common is their goal. As Cédric and Pierantonio [12] explain, the main task of matching algorithms is to *"[...] consider all the elements of both versions of the model and decide whether an element in the first version is the same as another one in the second version"*. They also explain that finding an element in the second model that is most similar to the analyzing element in the first model is one of the most complex and time-consuming tasks when calculating a comparison model.

As part of this work several existing implementations of differencing algorithms have been analyzed and evaluated. The results are presented and discussed in the following sections.

3.2.1 3DM - a Three Way Differencing and Merging Algorithm

Lindholm [13] explains in his thesis a design and implementation of a three way differencing and merging algorithm, called 3DM, which stands for 3-way merging, differencing and matching. This algorithm provides 2-way and 3-way comparisons of XML files as well as a mechanism for synchronizing them by performing appropriate merge operations. The aim of the thesis was to design and implement a mechanism for synchronization of data residing in different sources.

In order to reach this goal 3DM analyses the following operations applied to the data tree of an XML file by a user:

- **Insert**: A new node was inserted into the tree.
- **Delete**: An existing node or subtree was deleted from the tree.
- **Update**: The content of an existing node changed.
- **Move**: A node has moved to another position within the tree.
- **Copy**: A node has been duplicated and inserted at another position of the tree.

Since Lindholm's implementation is a prototype there are some limitations in the implementation of this algorithm. One of them, which is also important for our use cases (see Section 4.4.2), is that the algorithm is not aware of the XSD schema, when calculating the differences and performing the merge operations. Thus, after merging XML documents the resulting document can no longer be guaranteed to contain only valid content structured according to its XSD schema.

3.2.2 EMF Compare

The EMF Compare website [14] gives a good overview of the EMF Compare framework. It states:

"EMF Compare provides comparison and merge facility for any kind of EMF Model. In a nutshell this project provides:

- *a framework you can easily reuse and extend to compare instances of your models*
- *a tool integrated in the Eclipse IDE to see the differences and merge them*

It includes a generic comparison engine and the ability to export differences in a model patch. It is integrated with the Eclipse Team API meaning that it enables collaborative work on models using CVS, SVN and GIT".

Additionally they describe the framework, which has the properties described below.

Extensibility and customization. This means the framework provides mechanisms, which allow the definition and implementation of custom-specific matching and differencing algorithms, if this is necessary. The user can put the focus on the implementation of the important parts such as the matching algorithm itself and trivial parts of the calculation such as parsing and converting the model are performed by the framework.

Scalability. This means the framework is able *"to compare models with millions of elements in a number of steps proportional to the number of differences".* For this purpose it loads only the important parts of the model, which makes it possible to calculate a comparison within an optimal amount of time and memory.

Integrability. This means that EMF Compare offers application programming interfaces (APIs) that facilitate comparing models stored in repositories.

As described in the previous sections matching algorithms for model differencing can follow several approaches. As stated in [11] the generic matching algorithm of the EMF Compare framework uses a similarity-based matching approach, which is basically based on statistics and heuristics. Additionally it analyses properties such as name, type, relations to other elements and the content of an element to find appropriate matches [12]. When using this generic approach, the calculation of the comparison model is performed without using a unique identifier and thus, the model elements are not required to contain such a feature. Nevertheless, the matching and differencing algorithms can be adapted and extended to use a unique identifier, when necessary for a certain model.

3.2.3 Discussion of Existing Model Differencing Algorithms

Two different existing approaches for differencing and merging models stored in XML documents have been presented in the sections above.

The advantage of Lindholms 3DM algorithm is that no conversions are needed to be applied to the XML data models. Thus, since no time-consuming preprocessing has to be performed, the algorithm operates very fast. The big disadvantage of this approach is the missing knowledge of the meta-model to the algorithm. Thus, when performing a merge process without knowing the meta-model, it is not guaranteed that the resulting, merged XML model is still compliant to the meta-model. This is the reason, why this approach is not suitable for our use cases.

The other solution presented above uses the EMF Compare framework to create comparison models. The advantage here is that this algorithm is aware of the meta-model and thus, it can perform a meta-model compliant merge process. The disadvantage of this approach is the need of conversion processes for the meta-models and data models. In dependence of the model size, this process can be very time-consuming. The converted meta-model can be stored and reused, but each time the XSD meta-model changes, the conversion process needs to be executed again. Figure 4 illustrates this process.

3.3 Requirement Tracing and Referencing in Documents

Tracing and referencing of design artifacts in documents is important for the development of secure smart cards, especially for the documentation of the design and the security evaluation.

Traceability denotes *"[...] documenting the relationships between layers of information - for instance, between system requirements and software design"* [15], which also includes links between artifacts in the source code and the related documentation.

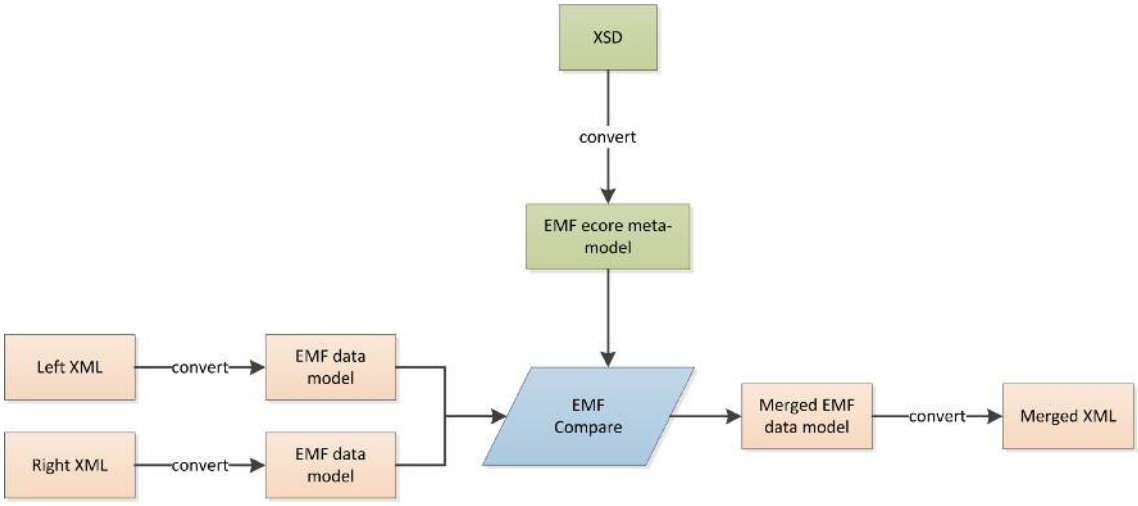

Figure 4: EMF Compare merging process.

The word *referencing* denotes in this context inserting the content of a design artifact, defined in a requirements management system (RMS), in the documentation.

Having a mechanism that supports the user in doing such tasks is very important to avoid inconsistency and redundancy of information when developing a product. The possibility to reference data e.g. from a RMS also facilitates product maintenance as changing a requirement in the RMS changes the requirement automatically in each document where it is referenced. This guarantees consistency in the documentation. Copying and pasting design artifacts within the documentation instead of using appropriate referencing mechanisms would increase the effort for maintenance enormously and therefore such a way of work must be strictly avoided. Doubling a design artifact in the documentation, doubles the effort for product maintenance, too. Tripling an artifact, triples the effort needed for maintenance, and so on.

Lots of technologies have been found during this research that deal with the traceability problem of design artifacts. In contrast, the number of technologies that offer support for referencing design artifacts and reusing their content in documents is still small.

In this section we discuss such technologies that allow tracing and referencing of requirements and specification artifacts in the documentation.

3.3.1 Dita XML as Technology for Referencing Design Artifacts in Documents

One of the technologies that puts the focus on the production and reuse of technical information is DITA XML, which is an abbreviation for Darwin Information Typing Architecture [16].

It *"is an XML based architecture for authoring, producing and delivering of technical information"* [16]. DITA XML puts the focus on a reuse by reference concept with the main aim to separate the content from its context. In other words this technology facilitates the reuse of independent text blocks in different contexts and documents. For this purpose the content needs to be divided into independent topics and can then be inserted in several contexts without the need of being rewritten. Thus, the entire documentation stays always maintainable, because a change of a single text block changes the text in all documents this block is referenced in. In contrast to the approach using copy and paste for inserting a certain text block multiple times in one or more documents, the maintenance effort for the documentation does not increase with the number of reused text blocks when following the described approach with DITA XML.

As stated in [17] DITA XML uses information types for describing the structure and semantics of a certain kind of content. As shown in Figure 5 for creating content the user can either choose from a set of predefined standard information types such as topics or concepts etc. or define its own information types. More specialized information types can be either derived from a standard type or developed completely from scratch.

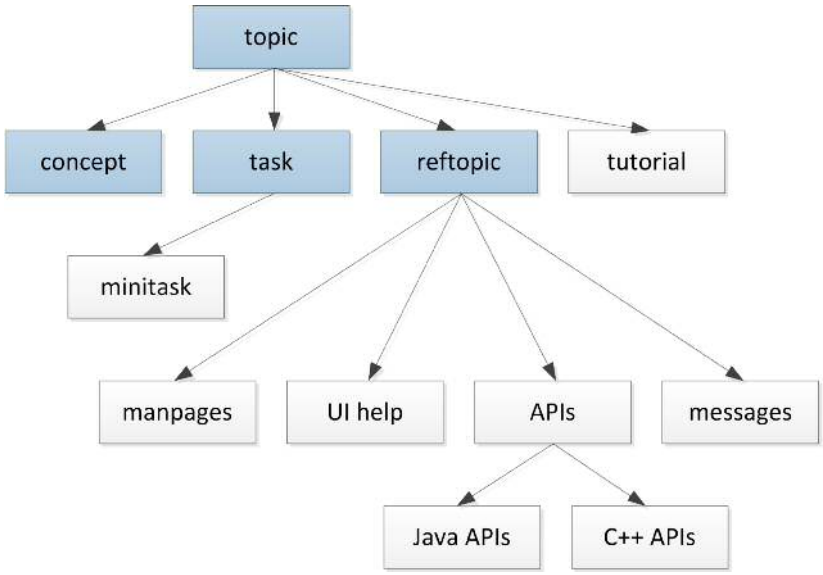

Figure 5: DITA XML information types [16].

This technology also allows to organize content in so-called maps. A map is a document, in which references to several content blocks are structured and organized. This allows managing related topics and putting them together in a single document.

3.3.2 DocBook as Technology for Referencing Design Artifacts in Documents

Jenkins and Heron [18] describe in their paper an approach for including model data in engineering and design documents by using ontologies and a technology called DocBook.

When using their workflow, for producing a document the following steps need to be performed. In the first step the model content is exported from an arbitrary requirements database. The exported data can either be stored in a proprietary or an open data format, but it must be human readable. This is necessary for being able to convert the obtained data to a common ontology, which is performed in the next step.

"An ontology is simply a set of classes, properties, and relationships that is useful for some domain of discourse" [18]. This kind of data representation is basically used to exchange data between several applications and services.

Then, the data is extracted from the ontology and converted into DocBook instances. DocBook is an *"[...] open, standards-based interchange format for technical documents [...]"*. It is *"[...] a Document Type Definition (DTD) for the Standard Generalized Markup Language (SGML) or the eXtensible Markup Language (XML)"* [18]. This has the advantage that XSLT transformations can easily applied to such documents to convert them to another file format. XSLT is a language that allows to apply transformations to XML documents and convert them to another format by applying user-defined rules [19].

For defining the structure of the final document, the template engine of the Ruby[3] standard library is used. This is necessary to define, which parts of the document contains informal text in natural language, and the locations where the formal text blocks, such as requirements from the database, are inserted.

Finally, a DocBook generator is used to produce the final DocBook instance.

They see lots of advantages by following such an approach:

1. Less effort for producing engineering documentation.
2. Avoiding of inconsistency between a requirements database and the according documentation.
3. Usage of a file format, that is simple, stable and well documented, which makes sure the documentation is still readable in the future.

[3] https://www.ruby-lang.org

3.3.3 ADE Documentation Framework

During a previous work [20] the so-called Advanced Development Environment (ADE) has been developed, which is a framework that uses LaTeX as technology for producing engineering documents. LaTeX is a text processing software based on a markup language where text and design are separated from each other. This technology also allows the user to define its own macros, which are prefabricated, user-defined text blocks that can be inserted by a self-defined command [21].

This functionality is used here to create specific commands for referencing and inserting different kinds of requirements and specification artifacts. ADE generates these commands automatically out of the specification of a dedicated product. This enables to use them in the according documents.

The documentation engine supports different formats in which specification can be delivered to the engine:

- **Normalized XML:** A special XML format for storing requirements and specification artifacts.
- **IBM Rational DOORS:** A commercial requirements management system.
- **Microsoft Excel:** Specification stored in special structured Excel tables.
- **CSV:** Specification stored in simple text files, where columns are separated by semicolons and data records are separated by line breaks.

Figure 6 gives a rough overview of the data flow during the production of the documentation. It shows how data and figures from different sources flow into the final document. For this purpose specification data is converted into a so-called normalized XML format. This is a special structured XML format for storing design artifacts. Based on this data, LaTeX macros are generated, which allow the developer to insert and reference any design artifacts by using the according commands. The implementation of the framework is based on several scripts that are executed by a *make*[4] flow. This ensures that project dependencies between modules are resolved in the right order.

Table 3 shows examples of commands that can be used for inserting formal parts (requirements, specification) and combine them with informal text in a document. Basically, for each

[4] http://www.gnu.org/software/make

specification type there exists a set of commands that allow inserting requirements and specification artifacts into the documentation.

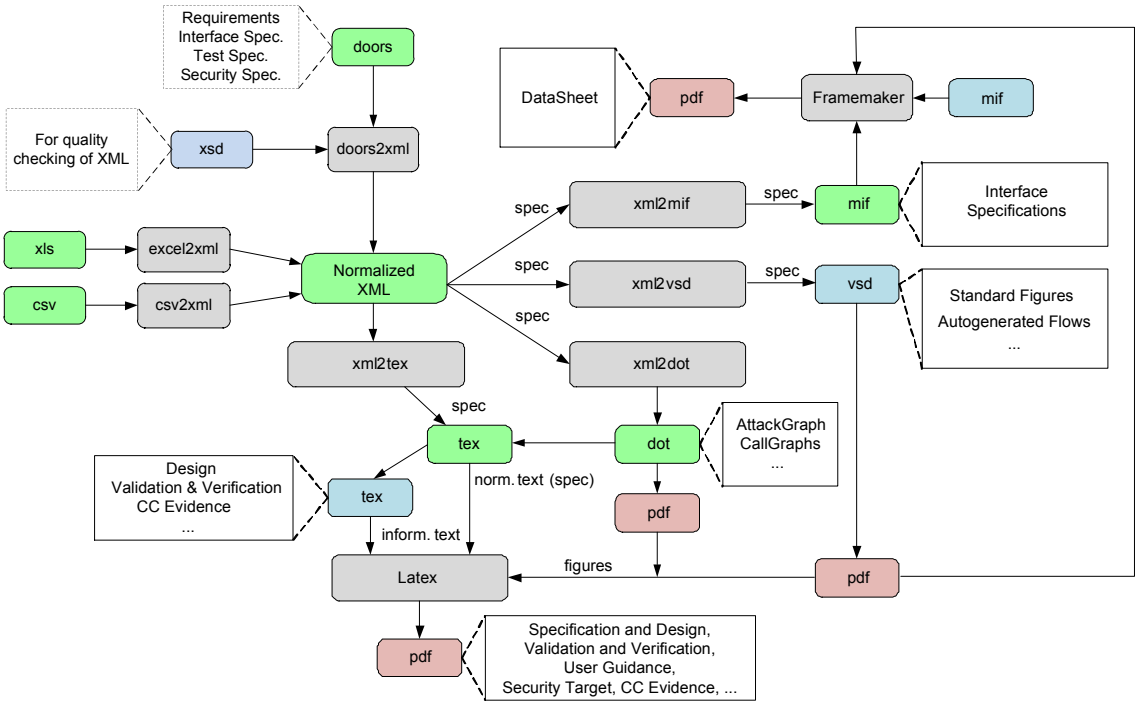

Figure 6: Data flow for documentation engine [20].

A similar approach exists for writing data sheets with *Adobe FrameMaker*[5]. The documentation framework also supports the generation of figures for some selected specification types. Thus, for example the communication between a smart card and the corresponding reader can be visualized automatically based on the specification. Moreover, the framework also can generate *dot graphs*[6]. This facilitates for instance the insertion a figure in a document containing the hierarchical structure of the scopes for a dedicated product. This entire process and also the generation of the final documents are managed by the documentation engine.

[5] http://www.adobe.com/at/products/framemaker.html

[6] http://www.graphviz.org

Listing 1 shows a simple example of a LaTeX document (e.g. design document) in which the description of the requirement number 21 of scope *Bootflow* has been inserted.

```
\begin{document}

This is a sample informal text in the documentation. And this is requirement number 21 of scope Bootflow:
\RSTxt{Bootflow}{21}

\end{document}
```

Listing 1: Simple example for displaying a requirement in the documentation.

In order to guarantee the documentation engine being able to deal with different kinds of specification, it implements a plug-in interface, which allows the user to extend the core engine by defining new specification types. A specification type is implemented as a XSD meta-model that defines the structure of the normalized XML used to store the according specification.

3.3.4 Discussion of the Existing Technologies

As discussed above, DITA XML is a technology that facilitates the reuse of content in documents. This can also be used to reference and reuse design artifacts stored in a RMS in the documentation. For this purpose the documentation must be written in DITA XML and design artifacts would need to be exported and converted to this format and organized in suitable DITA maps. Each artifact would need to be stored in a predefined information type and related artifacts would need to be structured in appropriate DITA maps. With this setup it is possible to create design documentation containing formal and informal content. The formal content exported from the RMS can then be referenced by using the techniques described above.

What needs to be emphasized is that the problem of the low performance of the data export as described in Section 2.3 cannot be solved with this approach. A data export and conversion process is still necessary for the production of the documentation when following this approach.

Type	Command	Description
Reference	\RSReq{$Scope}{$Req}	Creates a hyperlink to a requirement with respect to the given scope and requirement parameter
Reference	\RSAsm{$Scope}{$Assumption}	Creates a hyperlink to an assumption with respect to the given scope and assumption parameter
Reference	\RSAssert{$Scope}{$Assert}	Creates a hyperlink to an assertion with respect to the given scope and assert parameter
Text	\TxtRSReq{$Scope}{$Req}	Displays the text of the requirement with respect to the given scope and requirement parameter
Text	\TxtRSAsm{$Scope}{$Assumption}	Displays the text of the assumption with respect to the given scope and assumption parameter
Text	\TxtRSAssert{$Scope}{$Assert}	Displays the text of the assertion with respect to the given scope and assertion parameter
List	\ListRSGroup{$Scope}{$Group}	Displays all requirements located in the given scope and group
List	\HListRSGroup{$Scope}{$Group}	Displays all requirements located in the given scope and group including all headings
Structure	\TreeRS{$Scope}	Displays all requirements of the given scope (recursively)
Structure	\TreeRSGroup{$Scope}{$Group}	Displays all requirements of a given scope and group (recursively)
Structure	\HTreeRS{$Scope}	Displays all requirements of the given scope including the headings (recursively)
Structure	\HTreeRSGroup{$Scope}{$Group}	Displays all requirements of the given scope and group including the headings (recursively)

Table 3: Selection of possible commands to insert or reference requirements in the documentation.

Chapter 4

Concept and Design

This chapter describes the concept and design of the optimized development process. It gives an overview of its goals and requirements and a detailed explanation of its structure and architecture.

4.1 Goals

From the problem definition in Chapter 2 a set of requirements and goals has been derived for the new, optimized development process, which is discussed in the following.

[Goal 1]: Unified process for development and security evaluation.

A unified process for development and security evaluation facilitates the linking of artifacts between the development process and the security evaluation process. This supports the user in creating the evidence for product security. This is part of the security evaluation and thus necessary for the certification of the product.

[Goal 2]: Process for agile development and security evaluation.

The focus for this process is to depict agile development [22] and security evaluation [23]. Highsmith and Cockburn [24] describe the advantage of agile development in the following way:

"Working code tells the developers and sponsors what they really have in front of them - as opposed to promises as to what they will have in front of them. The working code can be shipped, modified, or scrapped, but it is always real". And further they describe this method as approaches that *"[...] recommend short iterations in the two- to six-week range during which the team makes constant trade-off decisions and adjusts to new information".*

For both the development of the product itself and for the security evaluation an agile approach has several advantages. Breaking a system down to a module level and certifying each module by following an agile approach makes it easier in the final development phase to get the finished product certified. The concept of agile development is depicted in Figure 7. It shows the parallel development of specification, design, implementation, testing and documentation for each scope in an iterative way.

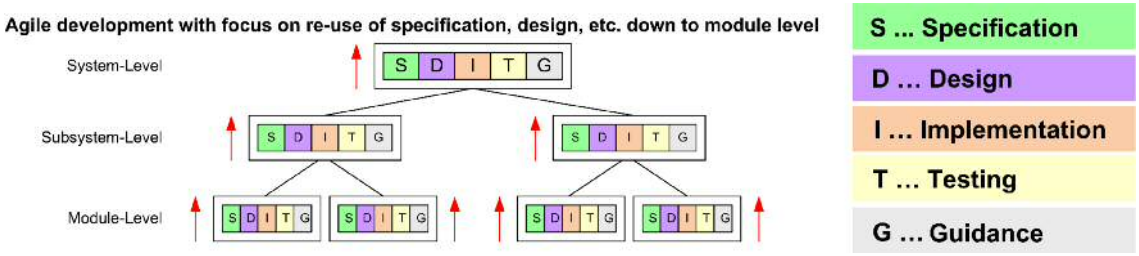

Figure 7: Concept of agile development.

[Goal 3]: Linking, tracing and referencing of requirements.

A linking mechanism is needed as a basis for tracing of design artifacts.

Traceability of requirements and specification artifacts supports analyzing a product in case of a change request. The tracing between artifacts of the development process and artifacts of the security evaluation process helps to produce an evidence that the product is secure.

A mechanism for referencing design artifacts in the documentation is necessary to avoid copy and paste operations for inserting design artifacts into the documentation. This again avoids inconsistency and decreases the effort for product maintenance.

[Goal 4]: Support managing of requirements.

During the analysis of the development process, several problems have been identified related to the RMS DOORS. Thus, one of the goals is to find an appropriate RMS that fulfills the according requirements listed in Section 4.2.1.

[Goal 5]: Differencing and synchronizing of design models.

In order to be able to reuse interface specification by following a *single-point-of-source* approach, it is necessary to synchronize interfaces defined in the source code with the according design artifacts in the RMS. This facilitates referencing and tracing of interface specification in the documentation.

Furthermore, a generic synchronization mechanism allows to merge similar models. This can be used to support teamwork. When two developers work on the same model, all changes need to be merged to get a resulting model containing all changes. See Section 4.4.2 for a detailed description of these use cases.

Table 4 summarizes all identified goals for the optimized development process and lists the problems they cover.

Goal	Description	Covers
Goal 1	Unified process for development and security evaluation	Problem 1
Goal 2	Process for agile development and security evaluation	Problem 5 Problem 7
Goal 3	Linking, tracing and referencing of requirements	Problem 2
Goal 4	Support managing of requirements	Problem 3 Problem 4 Problem 5 Problem 6 Problem 7 Problem 8
Goal 5	Differencing and synchronizing of design models	Problem 5 Problem 7

Table 4: Summary of the goals for the optimized development process.

Thus, the development process for secure smart cards is going to be optimized in two dimensions:

- Tooling and environment used for development, documentation, implementation and security evaluation. The according optimization steps are described in Section 4.4.
- Structure of the development process and how different specification artifacts are linked with each other. The according optimization steps are described in Section 4.5.

4.2 Requirements

The requirements for the tooling environment for the optimized development process consider also the agile development. They can be divided into requirements for the following parts: specification repository, interface synchronization, testing and verification, project and product configuration, documentation, look and feel. The requirements for each of these parts are discussed in the following.

4.2.1 Specification Artifact Repository

[Req. 1] The specification repository should structure requirements and specification artifacts in reusable intellectual property blocks (IP blocks). These are reusable units of logic used for the development of ICs.

[Req. 2] Furthermore, this repository should contain all requirements and specification artifacts necessary for the development and security evaluation process.

[Req. 3] For the purpose of requirements traceability it should be possible to create different kinds of links between requirements and specification artifacts. Since such links might only be valid for a certain product, they should be stored in a product configuration and not directly in the IP block. The types of links that must be supported are listed in Table 5.

[Req. 4] An option to create new and manage existing branches of scopes and IP blocks should be implemented. This is necessary to be able to define IP blocks that have almost the same functionality, but differ in some details from each other. Additionally a versioning mechanism must ensure being able to track and reconstruct changes of an IP block over time.

[Req. 5] A generic approach should allow the user to define arbitrary specification types. These are meta-models that describe the structure of a requirement or a certain type of specification such as test specification or interface specification. They must be customizable and extendable by the user.

[Req. 6] In order to support the user by creating new data items and to reduce the error rate, a suitable application must be implemented, which allows the user to operate with any kinds of requirements and specification artifacts. Therefore the data shell be visualized in tabular form. Additionally a data validation mechanism must ensure all predefined constraints to be satisfied.

[Req. 7] The system should support multi-user access, which means that a mechanism must ensure that multiple users can work on a module concurrently.

[Req. 8] A baselining mechanism should prevent a requirement or specification artifact from being modified after a product using this artifact is already brought to market. Such a baselined specification should still be usable in a new product, but cannot be modified anymore.

Type of Link	Description
Satisfies	e.g. used to link design artifacts of different levels of abstraction
Refines	e.g. used to link refined attack artifacts or security architecture artifacts to each other
Tests	e.g. used to link test specification to the according design artifacts
Guides	e.g. used to link user guidance to the according design artifacts
Specified by	e.g. used to link interface specification to the according design artifacts
Attacks	e.g. used to link attacks to a security architecture
Covers	e.g. used to link a security architecture back to a certain attack
Realizes	e.g. used to link security functional requirements to its realization, namely the according security architecture

Table 5: Overview of link types.

4.2.2 Interface Synchronization

[Req. 9] Hardware and software interfaces should be defined in a central specification repository. This makes it possible to reference them in documents for design and security evaluation.

[Req. 10] A tool should be implemented that generates C header files and Verilog files out of the according interface specification in the central specification repository. The C header files contain the firmware interfaces and the Verilog files contain the hardware interfaces. Furthermore, it should be possible to synchronize changes of interface definitions in the C header files and Verilog files with the according artifact in the central specification repository. This supports the agile development and prevents that developers must regenerate the header files after each source code modification manually.

4.2.3 Testing and Verification

[Req. 11] Since the specification repository should be used for all kinds of artifacts, software and hardware tests should also be defined there. Similar to the interface specification it must also be possible to generate basic structures of test implementations out of the according data objects in the specification repository. The definition of the test function and comments for each step in the test case should be inserted automatically in the according test implementation.

[Req. 12] A test report generator should be able to create reports that contain the test specification and their results in a single document. Such documents are then stored in the content repository.

4.2.4 Project and Product Configuration

[Req. 13] It should be possible to create a new project and define its structure including the components, scopes and subscopes.

[Req. 14] It should be possible to manage product configurations. A product configuration contains all information relevant for a dedicated product including its scope structure and configuration. A scope can contain either a simple link to an IP block in the specification repository or product dedicated content, which is then stored directly in the product configuration.

4.2.5 Documentation

[Req. 15] A suitable documentation engine should be implemented that allows generating parts of the documentation automatically by referencing requirements and specification artifacts in documents. Additionally the system should be able to print lists and tables of groups of related requirements or specification.

4.2.6 Look and Feel

[Req. 16] The graphical user interface for the overall system should be useable by intuition.

4.2.7 Relations of Problems, Goals and Requirements

Table 6 lists all requirements and the addressed problems and goals.

Goal	Covered problem	Addressed by requirement
Goal 1	Problem 1	Req. 3
Goal 2	Problem 5; Problem 7	Req. 7; Req. 10
Goal 3	Problem 2	Req. 9; Req. 12; Req. 15
Goal 4	Problem 3; Problem 4 Problem 5; Problem 6 Problem 7; Problem 8	Req. 1; Req. 2; Req. 3 Req. 4; Req. 5; Req. 6 Req. 7; Req. 8; Req. 11 Req. 13; Req. 14
Goal 5	Problem 5; Problem 7	Req. 10; Req. 11

Table 6: Relations of problems, goals and requirements

4.3 Background

This section contains descriptions of systems that have been implemented as part of earlier works and are reused in the scope of this work. It describes their integration and usage in the optimized development process.

Since this work has been done in cooperation with a company, there have been some constraints such as the reuse of the existing meta-models for specification artifacts, which are described below.

4.3.1 Meta-Models for Specification Artifacts

As stated in section 4.1 the process should support arbitrary specification types, which can be adapted as needed by the user. A specification type is a set of attributes that defines the structure information to be stored for a certain kind of specification. For the development of secure smart cards and the corresponding security evaluation at least the following specification types are necessary to define the functional requirements and specification:

Definitions (Def). This specification type contains basic definitions such as maximum supply voltage, layout area for a certain hardware module etc. for a project.

Requirements (RS). This specification type contains requirements for a certain scope.

Test Specification (TS). This specification type contains specification for a test case including all assumptions preconditions, test steps and test setups.

Test Results (TR). This specification type contains the result of an executed test case including its name, the version of the test, the execution date, the name of the test engineer and the verdict (pass, fail, not executed, waived or unknown).

Known Limitations (KL). This specification type describes for a test case with the verdict waived, why this test case did not pass.

Interface Specification (IS). This specification type describes an interface. Since the information objects that describe hardware and software interfaces differ from each other, for each kind of interface a specification type must be implemented.

User Guidance (UG). This specification type allows writing the user guidance in a structured way.

Furthermore it is necessary to define the requirements for the product from a security point of view. For this reason the following set of specification types is necessary:

Security Architecture (SA). This specification type contains information that describes the requirements for the product from a security point of view.

Attacks (AT). This specification type stores information about possible attacks for a certain IC.

The security evaluation process requires also a special set of specification types:

- Common Criteria Security Assurance Requirements (CC-SAR).
- Common Criteria Security Functional Requirements (CC-SFR).
- Common Criteria Security Objectives (CC-SO).
- Common Criteria Security Problem Definition (CC-SPD).

In order to get a closer view on how such a specification type is structured in detail, the meta-model of the RS specification type is used to describe this approach in a short example. Figure 8 shows the structure of the RS specification type, its elements and attributes. Table 7 contains a description of the according attributes.

Attribute	Description
ID	An automatically generated unique identifier for the artifact for referencing and traceability reasons
Label	A user defined identifier for the artifact
Status	Defines the status of the artifact: Draft, Rejected, Accepted, Modifying
Priority	The priority for the implementation of the addressed feature: must have, nice to have, negotiable, none
Satisfied	Defines if the requirement is satisfied by a lower level requirement: yes, not
Qualified	Defines if the requirement is verified by appropriate tests: yes, not
Text	The description of the requirement
Sources	Defines the origin of the requirement, the person or group of persons who raised the requirement
SourcesText	The original text of the defined sources
Rationales	Definition of the reason behind the requirement
SatisfactionArgument	An argument that the system can meet the defined requirement
QualificationArgument	Rationale why a certain test method has been selected for this requirement
Comments	Any other comments that need to be stored together with the requirement
Links	Links between the requirement and other design artifacts

Table 7: Description of the attributes used in the RS specification type.

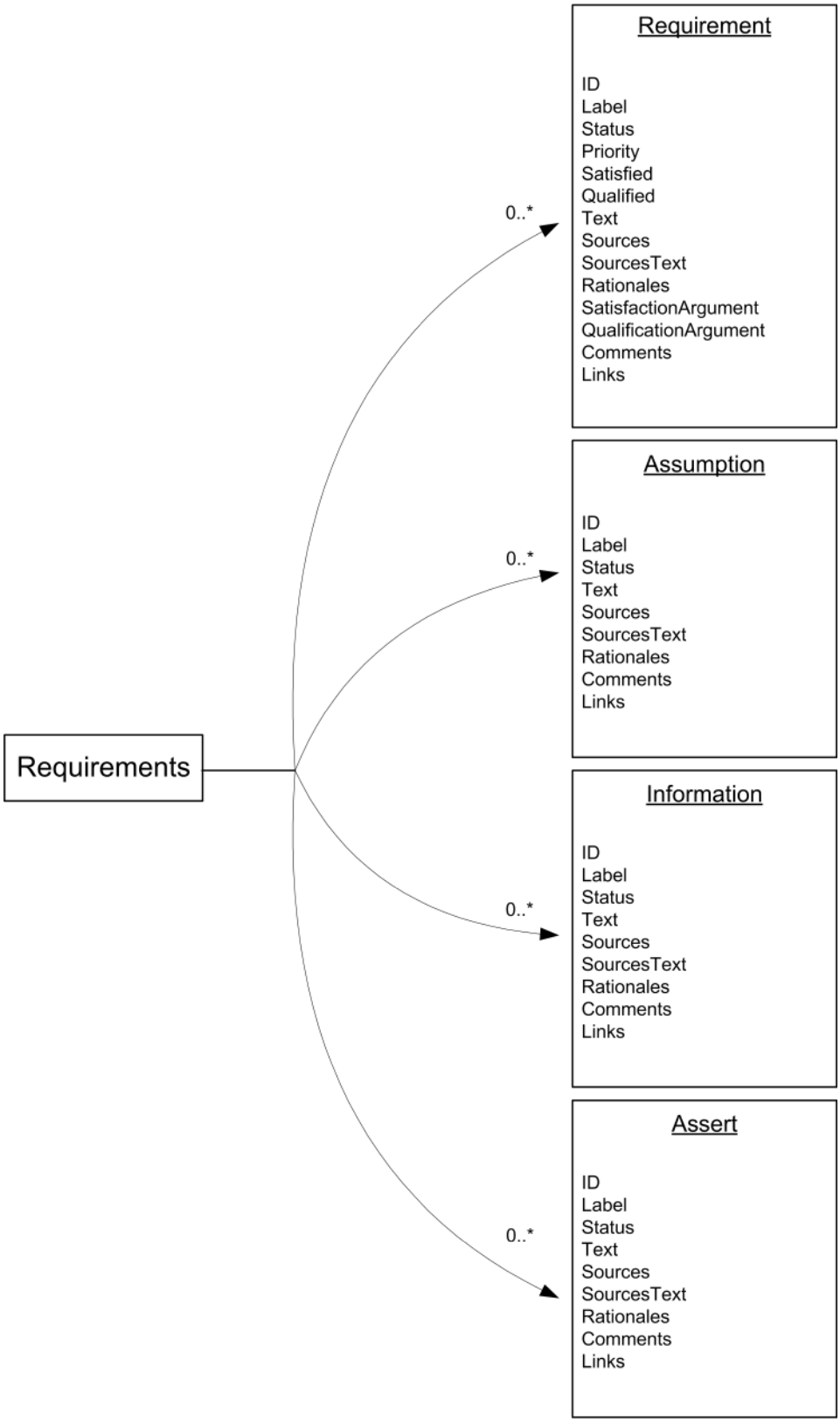

Figure 8: Structure of the specification type for requirements.

4.3.2 Documentation Engine

Overview

Another central part that must be considered in the design of the new development process is the creation of documents for design, security evaluation and user guidance. As stated in Section 4.1 a suitable documentation engine should implement at least the following features:

- Referencing of requirements and specification artifacts in documents.
- Displaying of requirements and specification texts in documents.
- Displaying of grouped lists of requirements and specification texts in documents.

The goal here is to follow a *single-point-of-source* concept, which means that requirements and specification artifacts shall be only defined on a single point. This is in our case the Requirements and Specification Management System (RSMS), described in section 4.4.1. As a result copy and paste operations can be avoided. This decreases redundancy and facilitates product maintenance enormously.

Concept

In Chapter 3 different existing approaches for creating engineering documents have been discussed. Due to the requirements for such a documentation environment, listed above and more detailed in Section 4.2, we decided to use the ADE framework [20] for producing documents.

ADE can be easily integrated into the optimized development process without the need of complex modifications and adaptations. Documents are created by using LaTeX in combination with this engine. This enables documents to contain references to single requirements and specification artifacts or a lists of them. Such references are then resolved by the documentation engine. It generates the final documents that are then again stored in the content repository.

Although this framework supports different kinds of data formats, in which design artifacts can be delivered, in the optimized, unified development process only the normalized XML format is used as input format for the documentation engine. The reason for this is that all design artifacts in the specification artifact repository are stored in the normalized XML format.

As described in Section 4.3.1 the user can define an arbitrary set of specification types by defining the according meta-models. These meta-models are also used by this documentation engine, which is necessary for dealing with references of design artifacts in the documentation.

4.4 Tooling and Environment of the Optimized, Unified Process

Figure 9 gives an overview of the tooling environment used for the optimized process. Tools are represented by rhombs, whereas artifacts such as documents, configurations and source code are depicted as rectangles. The roles of project members using a certain block are written in red letters. The following sections describe each single block of this environment.

4.4.1 Requirements and Specification Management System

The Requirements and Specification Management System (RSMS) is mainly used by project managers, system architects, test architects, security architects and security assurance engineers. They use this application to create and modify requirements and specification artifacts manually. Additionally, it offers the opportunity to create a comparison model of two design artifact models and to visualize their differences by analyzing the output of an external differencing engine. This functionality can be used to review the content changes of a certain module and to merge them. Since design artifacts are stored in a repository this functionality is also necessary to check the modifications of a module, when several team members work together on a single model concurrently.

According to the goals and requirements listed in Section 4.1 and 4.2 a requirements and specification editor needs to be implemented that supports users in working with requirements and specification artifacts stored in XML documents. These are located in the specification artifact repository.

Compared to the use of the commercial requirements management system IBM Rational DOORS this approach has several advantages. XML *"[...] is a simple text-based format for representing structured information: documents, data, configuration, books, transactions, invoices, and much more"* [25]. It is a non-proprietary and platform independent format. Since an export process from a proprietary database is no longer needed, it facilitates in our case the processing of this data for several purposes such as documentation and source code generation enormously.

The following paragraphs describe the architecture and design of the RSMS.

Figure 10 gives an overview of the architecture and data flow of the RSMS. The meta-models, which are descriptions of the structure of a model for a certain specification type, are stored as XML schemas.

"XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents" [26].

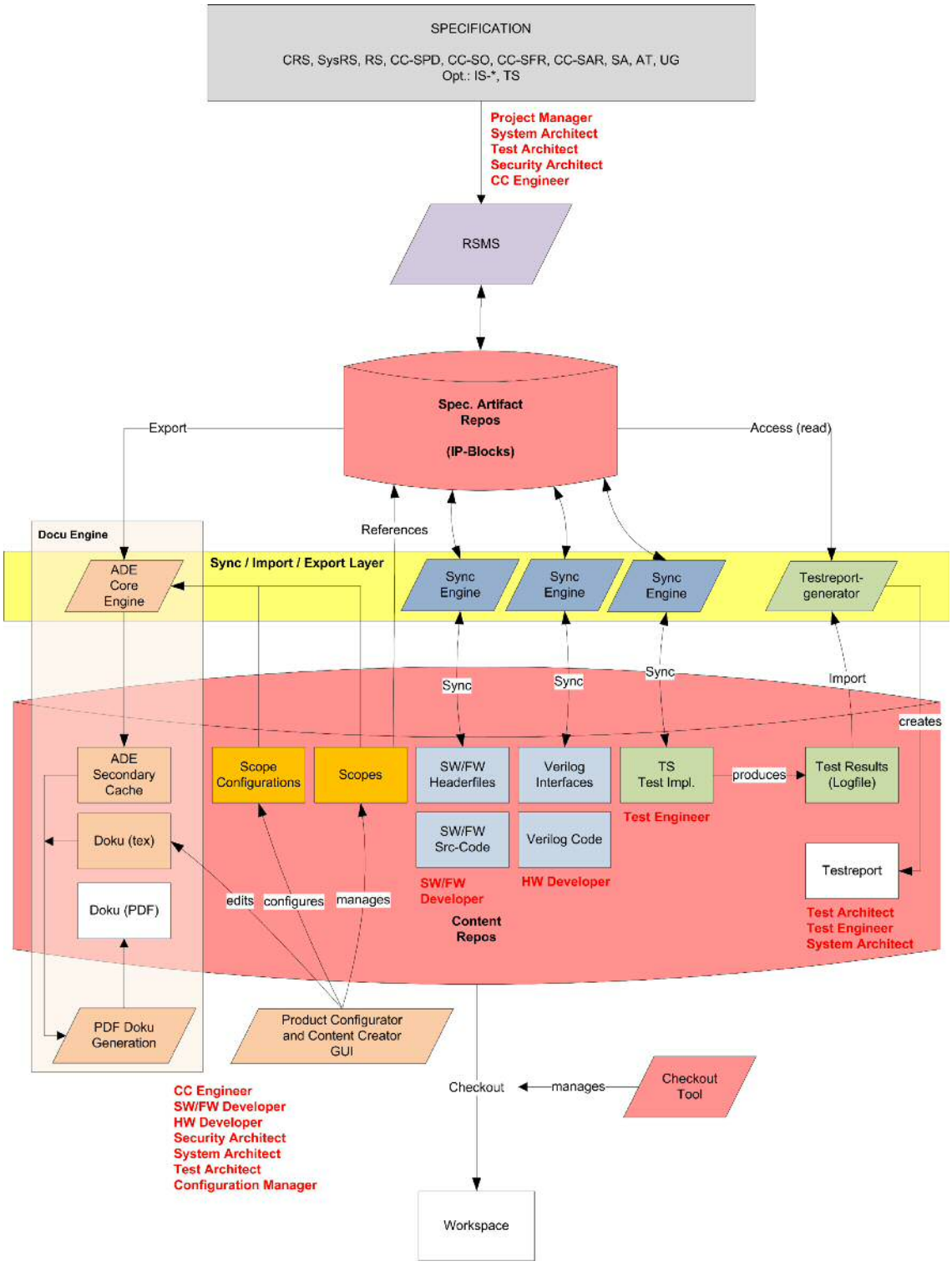

Figure 9: Overview of the environment for the optimized development process.

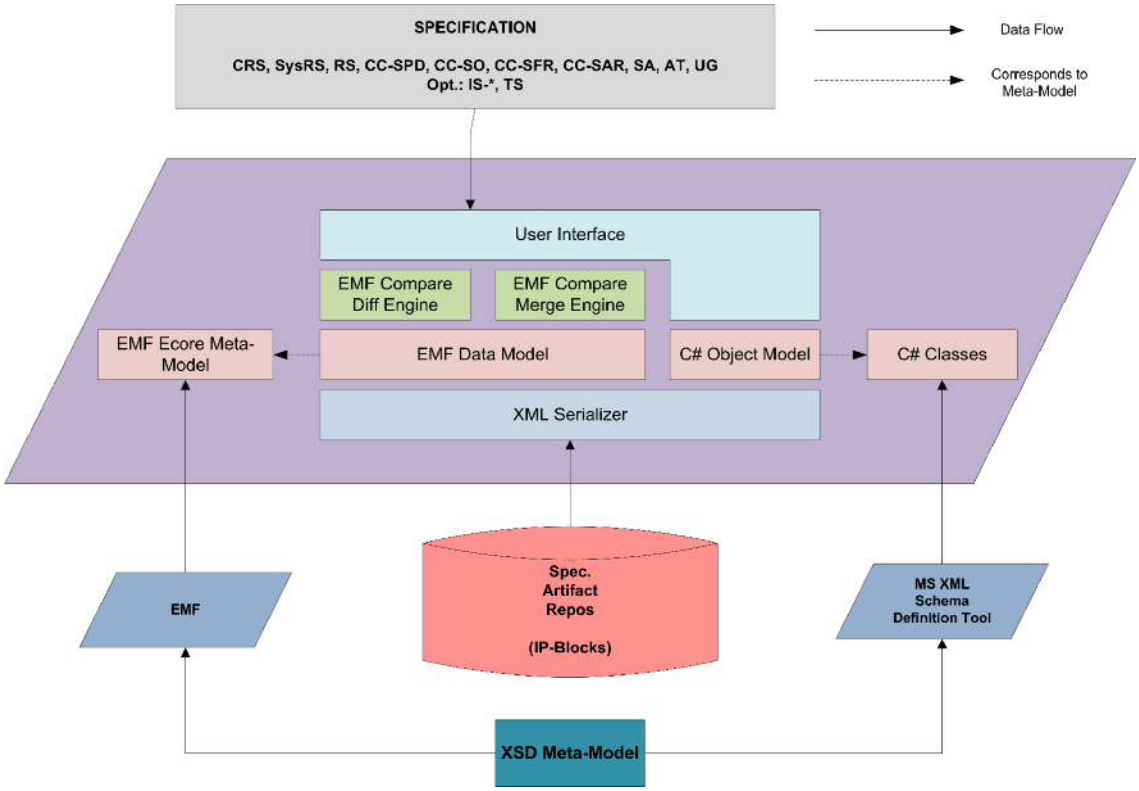

Figure 10: Overview of the architecture of the RSMS.

In the first step the XSD meta-model is imported to the Eclipse Modeling Framework (EMF) [14] and converted to an EMF ecore meta-model. Then the EMF and the generated ecore meta-models are used to read the XML specification artifacts and convert them to an ecore model, which is the basis for the EMF Compare based differencing engine.

In parallel the XSD meta-model is also used as input for the Microsoft XML Scheme Definition Tool [27] to generate the corresponding C# classes, which represent the structures of the meta-model in the C# world. These classes are then used together with the XML serializer to create a C# object model out of the XML specification artifacts.

As a result of these two processes, two types of models correspond to the same XSD meta-model: an ecore model and a C# object model. The C# object model is basically used for the operation with specification artifacts in the requirements and specification editor, whereas the ecore model is used for the differencing and merge engine based on the EMF. The reason for this way of work is that model comparison is not a trivial task and the EMF offers with its model comparison toolbox EMF Compare a powerful tool for such operations. In contrast, a simple data manipulation mechanism is easier to implement by using the C# object model, since after serialization of the object tree changes are automatically converted back to an appropriate XML format.

The differencing engine, which is based on the EMF Compare libraries shall be implemented as Java application and should use the generated ecore meta-models. In order to compare two specification models, the according XML files from the specification artifact repository need to be loaded into the EMF and converted to an ecore model. Then the EMF Compare based differencing engine creates a comparison model, which contains all detected differences. This is used as input for the user interface, which visualizes all model differences.

The engine shall be able to detect at least the following model differences:

- **Add**: A node has been added to the model.
- **Delete**: A node of the model has been deleted.
- **Update**: A property or content of an element has been changed.
- **Move**: A node has been moved to another (hierarchical) position within the model.

The implementation of the merge engine is not within the scope of this work, but for the sake of completeness it is also considered in the design and architecture.

The graphical user interface shall include a requirements and specification editor and is to be implemented as C# application. It should use the generated C# object model to operate with the data. The created object model is to be visualized and should offer the user an easy and intuitive way to work with design artifacts. Furthermore, an interface needs to be implemented that facilitates visualizing comparison models produced by the differencing engine.

The details of the implementation of all these tools and mechanisms are described in Chapter 5.

4.4.2 Import-Export-Sync Layer

The Import-Export-Sync Layer can be seen as the interface between the specification artifact repository and the content repository.

It mainly consists of three tools:

- Synchronization Engine.
- Documentation Engine (ADE Core).
- Test Report Generator.

The synchronization engine is used for generating and synchronizing interface and test specifications from different sources. Thus, it is possible to generate and synchronize parts of

the source code out of the according artifacts in the repository. The synchronization process can basically be divided into two separate processes:

- **Differencing:** Calculates a comparison model of two design artifact models.
- **Merging:** Merges the changes of the second model into the first model.

Thus, a differencing and merge engine needs to be implemented, which is described in the next section.

Differencing and Merge Engine

The Differencing and Merge engine (DM engine) is a central part when operating with requirements and specification artifacts from different sources. Possible technologies for implementing such an engine and their advantages and disadvantages have been analyzed and discussed in Section 3.2. Due to the need of the awareness of the meta-models, we decided to use the EMF Compare framework for the implementation of our DM engine.

The following three use cases have been identified, where a DM engine is needed:

- **Use Case 1:** Review of changes.
- **Use Case 2:** Synchronization of requirements and specification between different sources.
- **Use Case 3:** Teamwork; usage of a repository with multiple users.

Use Case 1. Figure 11 shows the review use case. Assume that a developer needs to modify the specification of a certain scope. For this purpose it is necessary to check out the according module from the specification artifact repository. After this step the module is going to be modified and checked in again. If an architect wants to review the changes, he can check out the newest revision and the base revision of the according module and use the DM engine to make the changes visible. In the next step the architect can decide which changes he wants to accept and which he declines by using the merge engine. This creates the new specification file, which then can be checked in again.

Use Case 2. Figure 12 shows the synchronization use case. Assume a software developer needs to modify a firmware interface. As we follow a single point of source approach for all kinds of specification to avoid redundancy (see section 4.3.2), interfaces must be changed in the specification artifact repository and the header files must be regenerated. To avoid this time consuming scenario the DM engine can be used to synchronize parts of the

specification between a source code file and the specification artifact repository. For this purpose the developer must convert the modified header file to an XML file, which shell be done by the interface synchronization engine. In the next step the according specification file must be checked out from the specification artifact repository. Both of these files are used as input for the DM engine, which analyses the differences and synchronizes the changes. The output of the DM engine is the synchronized specification module, which then can be checked in into the repository.

Use Case 3. Figure 13 shows the teamwork use case. It describes a situation where two persons work together on a single specification module of the specification artifact repository. For that purpose both team members must check out the specification module to be modified. When team member B has completed all the editing, the modifications are checked in again and it succeeds. When team member A is finished with all modifications and tries to check them in an error occurs. Due to the modifications of team member B the local copy of team member A is outdated and needs to be merged with the latest version from the repository. For this purpose the DM engine is used. It shows the differences between the local copy of team member A and the latest version in the repository containing the modifications of team member B, merges them into a new specification file, which then can be check in again.

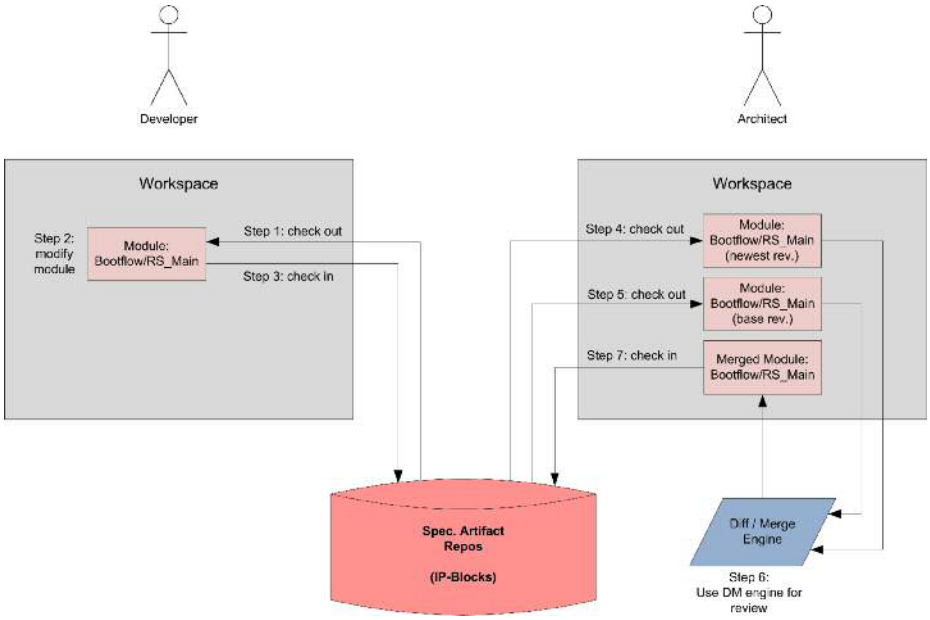

Figure 11: DM engine use case: review.

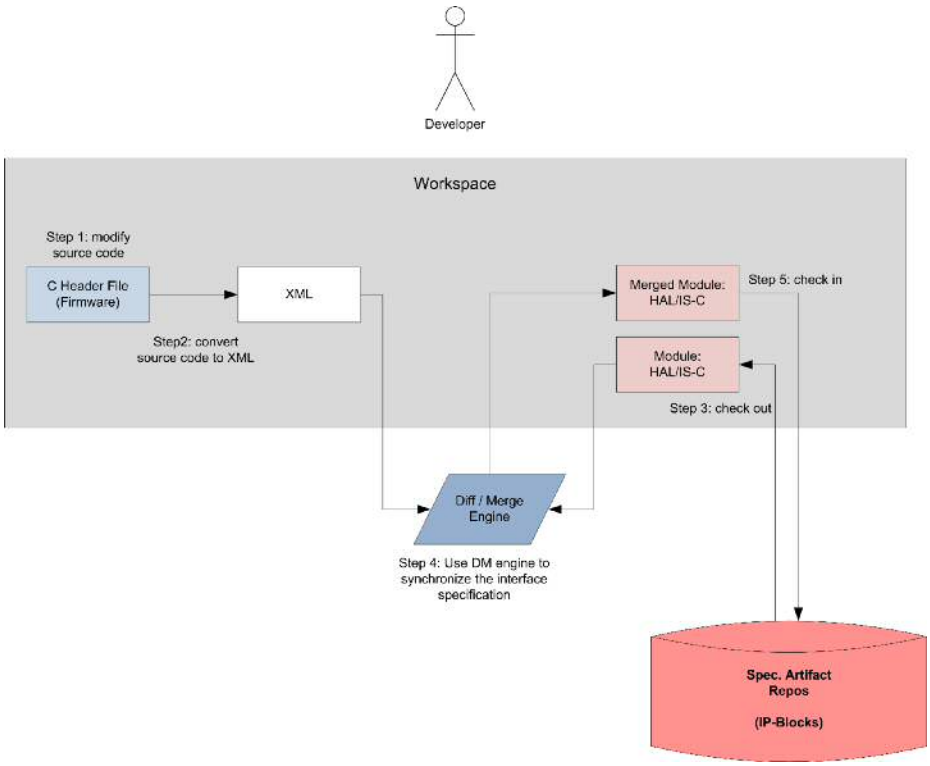

Figure 12: DM engine use case: synchronization.

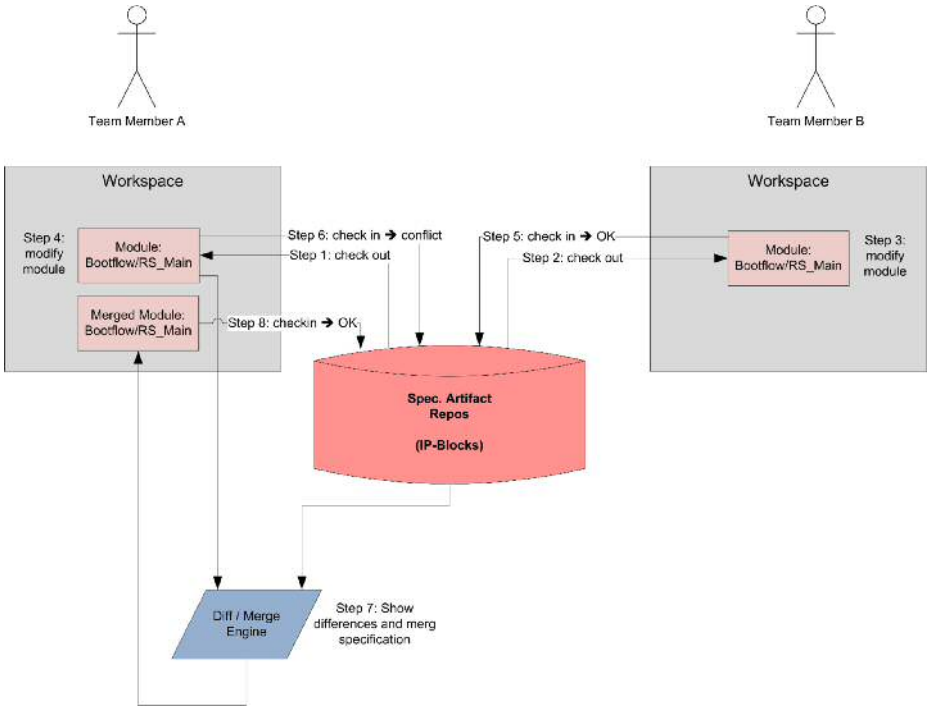

Figure 13: DM engine use case: teamwork.

Source Code and Test Case Generation and Synchronization

This section describes the synchronization process between the interface specification stored in the specification artifact repository and the actual interface definitions in the source code. In order to be able to synchronize such specification a number of preconditions must be fulfilled:

- Each interface definition in the source code must contain a comment to be able to identify the according artifact in the specification artifact repository.
- The comment must comply with a certain structure.
- The comment must at least contain the ID of the according interface specification in the specification artifact repository.

The minimum set of data in the source code that is to be synchronized with the artifacts in the RMS shall contain the following values:

- Name of the interface method.
- Name of each interface parameter.
- Type of each interface parameter.
- Type of the return value.

After triggering the synchronization process the following steps need to be performed. The C header file is parsed and the interfaces of the header file are matched to the according artifacts of the specification artifact repository. Then the resulting data is used to generate an XML file following the structure defined in the meta-model for interface specification. The XML file from the specification artifact repository complies with the same structure. Thus, it is possible to use the differencing and merge engine described above to calculate a comparison model, which is then used to merge the two XML files. The resulting data is then restored in the specification artifact repository. Additionally, this data is also used to refresh the interfaces in the source code. For this purpose the merged XML data is used to update the parsed interface specification of the header file. Finally the C header file is regenerated. As a result the C header file and the interface specification in the specification artifact repository are synchronized. Figure 14 illustrates the dataflow of the described process. The same principle applies to the generation and synchronization of test cases.

4.4.3 Configuration Management

This section describes the folder structure of the specification artifact and content repository and all details of the product and scope configuration. Following these structures is necessary for the smooth cooperation of the different systems and tools used in the development process.

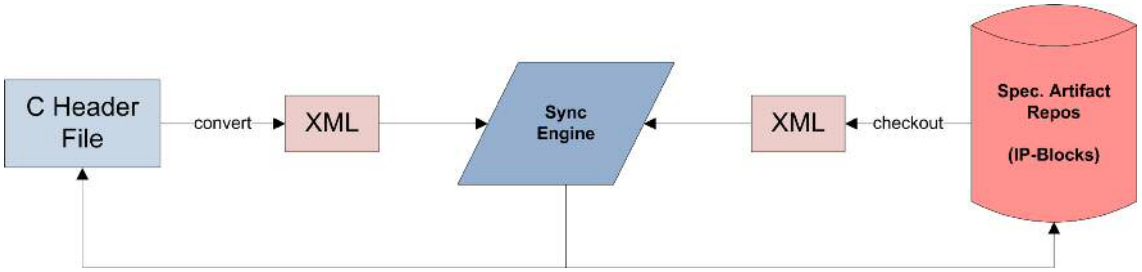

Figure 14: Data flow for source code synchronization.

Scopes

Figure 15 shows the folder structure of a single scope. All configurations and content associated with this scope are stored within this folder structure. The following list describes the content of each directory.

LatexDocuments. This directory contains the LaTeX projects for the documentation. A standard scope contains one project for specification and design documentation and another one for validation and verification. The corresponding *tex* files are located in these folders.

Specification. Contains the specification files for the scope. The specification is stored in the XML format.

Development. Depending on the type of the scope, it contains the software or hardware implementation (firmware modules, digital design etc.).

Secondary. The secondary directories are used as cache by the documentation framework (ADE) to store intermediates (e.g. parts of documents compiled as PDF or converted specification files etc.), used in the end to create the final documentation.

Subscopes. Contains all subscopes of the scope. A subscope has again the same structure as described here.

Basically, we distinguish between two different types of scopes in a product:

- Product dedicated scopes.
- Linked scopes.

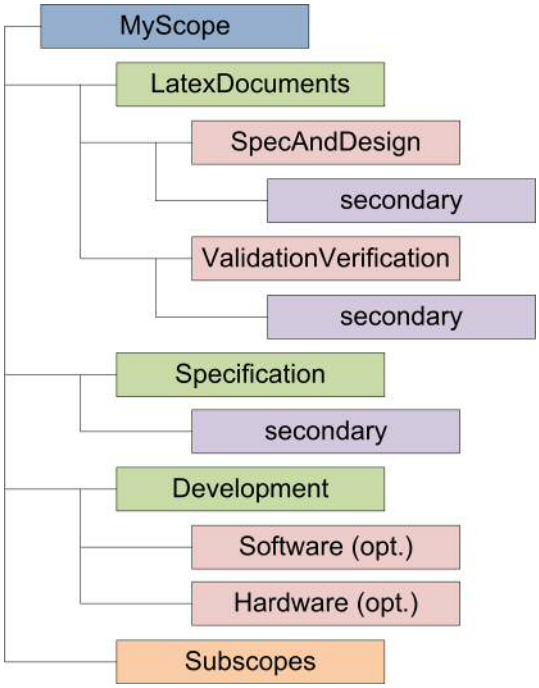

Figure 15: Folder structure of a scope.

Product dedicated scopes are product specific and a reuse of such a block in another product would not make sense. As a result these scopes are directly stored together with the product in the content repository.

A linked scope is a scope that contains the content of an IP block from the specification artifact repository. It is a more generic block that can be reused by several products. If a product uses such a scope the product configuration contains only a link to the according IP block in the specification artifact repository.

Specification Artifact Repository

Figure 16 shows the folder structure of the specification artifact repository. The specification artifact repository is the central location where scopes and IP blocks are stored. This repository has a simple structure, in which such blocks are grouped thematically. The IP library is basically divided into two parts: hardware IP blocks (HWIPs) and software IP blocks (SWIPs). Within these folders the IP blocks are again grouped thematically.

The repository depicts variants of a single scope as branches, whereas versions are represented by tags. It is the task of the RSMS to manage the scopes, variants and versions and thus to keep the branches and tags consistent.

Content Repository

Figure 17 shows the structure of the content repository. Whereas the specification artifact repository contains artifacts that can be used in multiple products, the content repository contains the configuration and content for a single dedicated product and its documentation and all development relevant files. Especially the following files are located in this repository:

Scope structure and scope configuration. This contains the structure of a dedicated product and links scopes to the according block in the specification artifact repository.

Firmware source code. This contains interface definitions (header files) and implementation of the firmware.

Digital design. This contains Verilog interfaces and descriptions of the digital design.

Test implementations. This contains the implementation of all product related test cases.

Test results and corresponding documentation. This contains the results produced by the test cases and test reports; a test report is a document that contains the specification of all test cases as well as their results.

Design documentation. Contains the design of the dedicated product to be developed.

Security evaluation documentation. Contains the documentation needed for the CC security evaluation process.

In the content repository single products are grouped within each product family. Only product dedicated scopes are depicted as folders. Linked scopes are only defined in the product configuration.

Product Configuration

A Product configuration defines the structure of a product and is defined in a so-called scope diagram XML file.

Listing 2 shows an example scope diagram for a simple product. This file contains the hierarchical structure of all scopes and their attributes used for a certain product. As discussed above, we have to distinguish between product dedicated scopes and linked scopes in a product. In case of a linked scope an URL attribute needs to be inserted that stores the link pointing to the address where the according block is stored in the specification artifact repository.

The product configuration and content creation GUI is used to create such product configurations and to setup its scopes. Additionally it offers an editor that supports the user in

creating LaTeX documentation. This editor is equipped with a auto-completion mechanism that makes it easy to insert or reference requirements or specification artifacts in the documentation.

```
<ScopeDiagram>
  <Scope Name="MyPlatform" Type="Mixed" Src=".">
    <Scope Name="Scope1" Type="Mixed" AbstractionType="Subsystem" Src="./Subscopes/Scope1">
      <Scope Name="Scope1_1" Type="Analog" Src="./Subscopes/Scope1/Subscopes/Scope1_1"
        Url="https://repository.mycompany.com/svn/IPLibrary/HWIPs/Coprocessors/
          Symmetric-Ciphers/AES/tags/release/v_1_0"/>
      <Scope Name="Scope1_2" Type="Analog" Src="./Subscopes/Scope1/Subscopes/Scope1_2"
        Url="https://repository.mycompany.com/svn/IPLibrary/HWIPs/Coprocessors/
          Symmetric-Ciphers/DES/tags/release/v_1_0"/>
      <Scope Name="Scope1_3" Type="Software" Src="./Subscopes/Scope1/Subscopes/Scope1_3"
        Url="https://repository.mycompany.com/svn/IPLibrary/SWIPs/OSs/SystemModeOS/tags/
          release/v_1_0" />
    </Scope>
  </Scope>
</ScopeDiagram>
```

Listing 2: Example of a scope diagram for a simple product configuration.

In order to check out a product from the repository into the local workspace, a special check out tool must be implemented. This tool populates the basic structure of a certain product from the content repository, analyses the links in the scope diagram and populates then the entire product according to its product configuration. This means that all links are resolved and the necessary IP blocks from the given repository locations are copied into the local workspace.

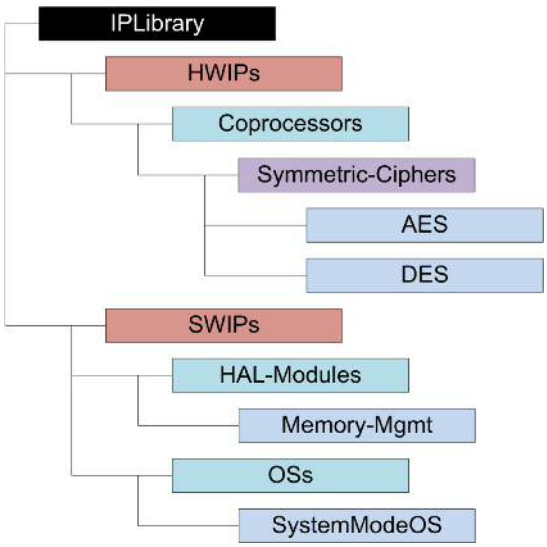

Figure 16: Folder structure of the spec. artifact repository.

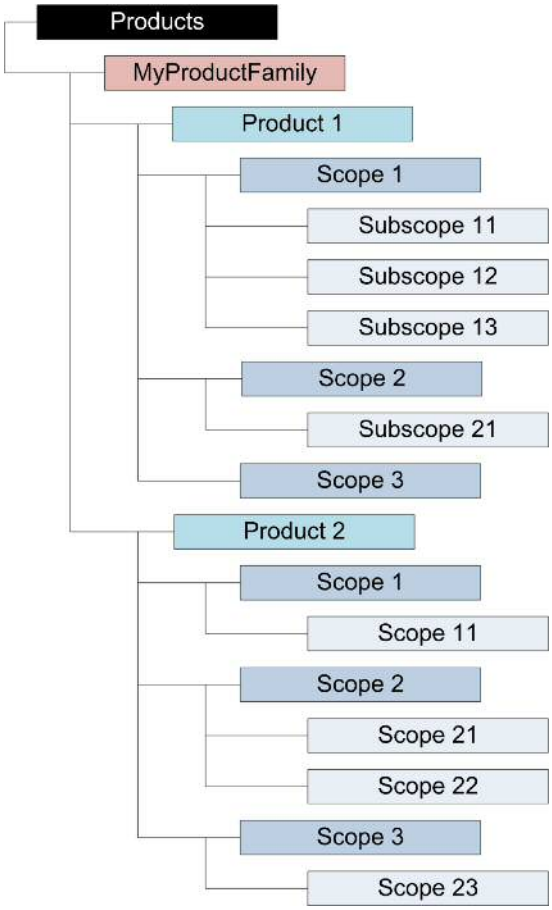

Figure 17: Folder structure of the content repository.

4.5 Structure of the Optimized, Unified Process

For the further optimization of the new process the aspects of security evaluation have been integrated into the development process. A single, unified process covering development and security evaluation is the result, which is shown in Figure 18.

Artifacts of the security evaluation process such as the security target are linked with the according customer requirements. Additionally, the requirements on system level are not only linked to the customer requirements, which they satisfy, but also to the security target. These links and mappings are necessary for the security evaluation and facilitate proving and analyzing the security of the product. Furthermore, the validation and verification plan (V&V plan) is now linked with the according test specification and this is again linked with the requirements they test on each level.

The design of a certain product is split up into several modules and submodules. These requirements are again linked to the according requirement on a higher abstraction level. Finally the implementation is linked with the design artifacts.

These merged process with all its links provided a full traceability between all artifacts created during the development of a product. The advantages of this approach are listed below:

- Simple impact analysis in case of change requests.
- Facilitates providing evidence of the security for a product.
- Linking information can be processed by a documentation engine and displayed automatically in documents.

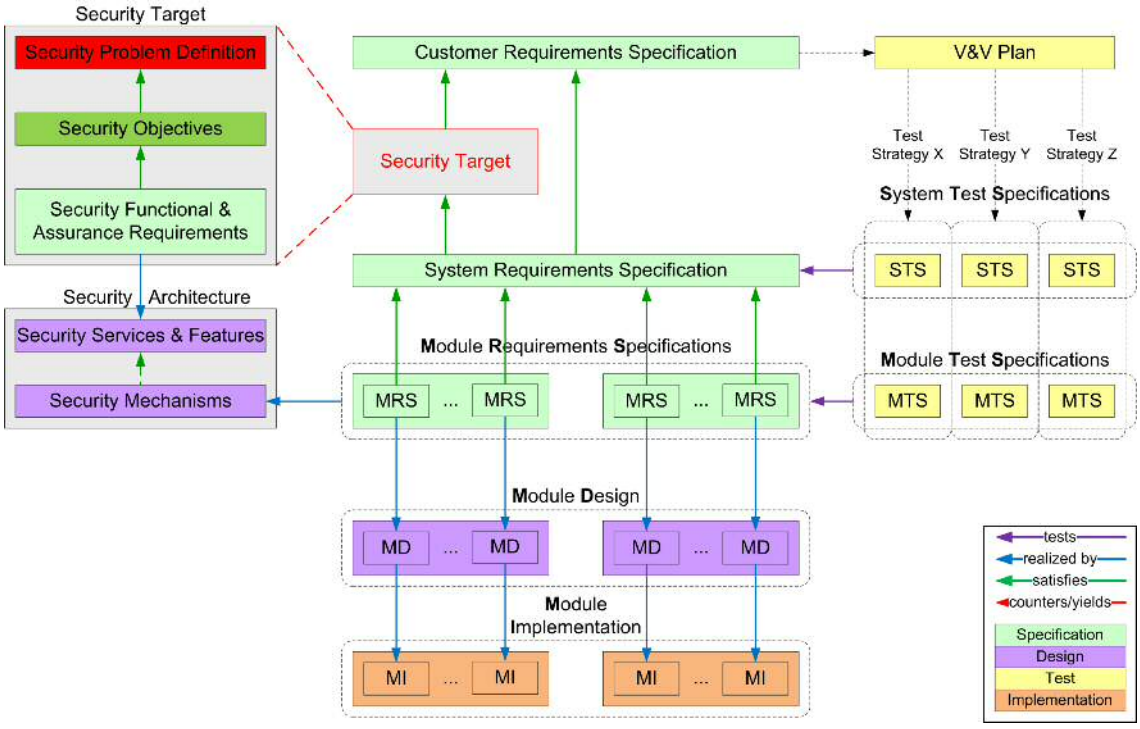

Figure 18: Overview of the optimized, unified development process [20].

Chapter 5

Implementation

This chapter describes the implementation of parts of the concept and design that have been within the scope of this work. Since the effort for implementing the entire process is very high, the focus was put on two parts: the implementation of a tool for requirements and specification management based on XML and the implementation of a differencing engine as basis for the use cases described in section 4.4.2.

This chapter is structured as follows. Section 5.1 describes the implementation of meta-models for design artifacts with a simple example. Such meta-models need to be implemented for each specification type needed to develop a product. They are used for the generation of the C# object model, which is described in Section 5.2, and the ecore data model, described in Section 5.3. Furthermore this chapter describes the access mechanisms for the C# object model, the implementation of the differencing engine for producing comparison models and the user interface. The latter allows the user operating with design artifacts and managing them. Furthermore it offers an interface for controlling the differencing engine and for visualizing the calculated comparison models.

5.1 Implementation of the Meta-Models for Specification Artifacts

As described in Section 4.3.1 the development process should support arbitrary specification types definable and extendable by the user. This section shows the implementation of a dedicated specification type for defining requirements. For the description of such meta-models the XML schema definition language (XSD) is used, which has the advantage to be platform-independent. Apart from that, one of the requirements for our implementation was to be compatible with existing specification meta-models, which are defined in XSD files.

In the optimized development process these meta-models are used by the following tools and applications:

- Requirements and Specification Editor.
- Differencing Engine.
- Documentation Engine.

For the requirements and specification editor, C# classes are created out of the XSD meta-model. The differencing engine uses a generated EMF ecore meta-model and the documentation engine uses the native XSD for the generation of the documentation. All these approaches have one thing in common: a *single-point-of-source* approach for the description of meta-models is followed. The XSD files contain the basic description of the meta-models and all other types of meta-models needed for different tools and applications are generated automatically out of the XSD files. Thus, in case the meta-model changes, the XSD files need to be adapted and all other models can be regenerated automatically. Listing 3 shows the XSD implementation of a specification type that describes the structure of requirements. The description of the meta-model can be found in Section 4.3.1.

5.2 Generation of the C# Object Model

For the generation of the specification meta-model in C# the Microsoft XML Schema Definition Tool (XSD tool) is used. The Microsoft Developer Network describes this tool as follows:

"The XML Schema Definition (xsd.exe) tool generates XML schema or common language runtime classes from XDR, XML, and XSD files, or from classes in a runtime assembly" [27]. Together with the *XmlSerializer* class provided by the .NET framework it can be used to create a binding between XML schema definitions and .NET class definitions. In other words, at development time C# classes can be created out of XSD schema definitions and at runtime arbitrary XML documents can be converted to objects by using the *XmlSerializer* class, which performs deserialization and serialization automatically [28].

Listing 4 shows a simple XSD schema and the according C# class generated by using the XSD tool.

This tool also has a number of limitations that need to be considered. One of them is that not all types and definitions supported by XSD documents can be handled by the XSD tool. A detailed documentation of this limitations can be found in the MSDN library [28]. Some of the limitations are also important for our implementation and must be considered in a proper way. To these belong for instance the interpretations of the *maxOccurs* and *minOccurs* attributes that

can be used within choice and sequence elements and define the minimum and maximum occurrence of an element or a sequence of elements on a certain place in the XML document.

A choice element contains a selection of elements, where only one of them can be inserted in the XML document. In contrast, a sequence element defines a certain number of elements that must be inserted in the defined order [29].

When using such attributes the XSD tool cannot fully convert the XSD scheme description to the according C# class definitions. A *maxOccurs* value of 0 is depicted as value of 1 and each value greater than 1 is depicted as unbounded. The *minOccurs* attribute defining the minimum occurrence of an element or a sequence of elements is completely ignored by the XSD tool and thus not depicted in the generated C# class.

When operating with the C# object model this leads to the problem that new inserted data items cannot implicitly be validated to be well formed according to its XSD scheme. Thus, due to these limitations we lose the implicit XSD schema check when inserting data objects.

```
<xs:simpleType name="typeEnumStakeholders">
  <xs:restriction base="xs:string">
    <xs:enumeration value="TBD"/>
    <xs:enumeration value="N/A"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="RS">
  <xs:complexType>
    <xs:sequence>
      <xs:choice maxOccurs="unbounded" minOccurs="0">
        <xs:element name="Req" type="tns:typeReqElements" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="Assumption" type="tns:typeAssumptionElements" maxOccurs="unbounded"
          minOccurs="0"/>
        <xs:element name="Info" type="tns:typeInfoElements" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="Assert" type="tns:typeAssertElements" maxOccurs="unbounded" minOccurs="0"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="typeReqElements">
  <xs:sequence>
    <xs:element name="Text" type="xs:anyType" maxOccurs="1" minOccurs="1" />
      <xs:element name="Sources" maxOccurs="1" minOccurs="0" />
      <xs:element name="SourcesText" type="xs:anyType" maxOccurs="1" minOccurs="0" />
      <xs:element name="Rationales" type="xs:anyType" maxOccurs="1" minOccurs="0" />
      <xs:element name="SatisfactionArgument" type="xs:anyType" maxOccurs="1" minOccurs="0" />
      <xs:element name="QualificationArgument" type="xs:anyType" maxOccurs="1" minOccurs="0" />
      <xs:element name="Comments" type="xs:anyType" maxOccurs="1" minOccurs="0" />
      <xs:element ref="ade:NeighborhoodLinks" maxOccurs="1" minOccurs="0" />
  </xs:sequence>
```

```
  <xs:attribute name="ID" type="xs:integer" use="required" />
  <xs:attribute name="Label" use="optional" />
  <xs:attribute name="Status" use="required" type="ade:typeEnumStatusOfDefinition" />
  <xs:attribute name="Priority" use="optional" type="ade:typeEnumPriorities" />
  <xs:attribute name="Satisfied" use="optional" type="ade:typeEnumSatisfaction" />
  <xs:attribute name="Qualified" use="required" type="ade:typeEnumQualification" />
</xs:complexType>

<xs:complexType name="typeAssumptionElements">
  <xs:sequence>
    <xs:element name="Text" type="xs:anyType" maxOccurs="1" minOccurs="1" />
    <xs:element name="Sources" maxOccurs="1" minOccurs="0" />
    <xs:element name="SourcesText" type="xs:anyType" maxOccurs="1" minOccurs="0" />
    <xs:element name="Rationales" type="xs:anyType" maxOccurs="1" minOccurs="0" />
    <xs:element name="Comments" type="xs:anyType" maxOccurs="1" minOccurs="0" />
    <xs:element ref="ade:NeighborhoodLinks" maxOccurs="1" minOccurs="0" />
  </xs:sequence>
  <xs:attribute name="ID" type="xs:integer" use="required" />
  <xs:attribute name="Label" use="optional" />
  <xs:attribute name="Status" use="required" type="ade:typeEnumStatusOfDefinition" />
</xs:complexType>

<xs:complexType name="typeInfoElements">
  <xs:sequence>
    <xs:element name="Text" type="xs:anyType" maxOccurs="1" minOccurs="1" />
    <xs:element name="Sources" maxOccurs="1" minOccurs="0" />
    <xs:element name="SourcesText" type="xs:anyType" maxOccurs="1" minOccurs="0" />
    <xs:element name="Rationales" type="xs:anyType" maxOccurs="1" minOccurs="0" />
    <xs:element name="Comments" type="xs:anyType" maxOccurs="1" minOccurs="0" />
    <xs:element ref="ade:NeighborhoodLinks" maxOccurs="1" minOccurs="0" />
  </xs:sequence>
  <xs:attribute name="ID" type="xs:integer" use="required" />
  <xs:attribute name="Label" use="optional" />
  <xs:attribute name="Status" use="optional" type="ade:typeEnumStatusOfDefinition" />
</xs:complexType>

<xs:complexType name="typeAssertElements">
  <xs:sequence>
    <xs:element name="Text" type="xs:anyType" maxOccurs="1" minOccurs="1" />
    <xs:element name="Sources" maxOccurs="1" minOccurs="0" />
    <xs:element name="SourcesText" type="xs:anyType" maxOccurs="1" minOccurs="0" />
    <xs:element name="Rationales" type="xs:anyType" maxOccurs="1" minOccurs="0" />
    <xs:element name="Comments" type="xs:anyType" maxOccurs="1" minOccurs="0" />
    <xs:element ref="ade:NeighborhoodLinks" maxOccurs="1" minOccurs="0" />
  </xs:sequence>
  <xs:attribute name="ID" type="xs:integer" use="required" />
  <xs:attribute name="Label" use="optional" />
  <xs:attribute name="Status" use="required" type="ade:typeEnumStatusOfDefinition" />
</xs:complexType>
```

Listing 3: Example of a fully implemented specification type for requirements.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://example.org/" xmlns=http://example.org/ elementFormDefault="qualified">
   <xsd:element name="complexInstance" type="MyComplexType"/>
   <xsd:element name="field1" type="xsd:string"/>
   <xsd:element name="field2" type="xsd:string"/>
   <xsd:element name="field3" type="xsd:string"/>

   <xsd:complexType name="MyComplexType">
      <xsd:all>
         <xsd:element ref="field1"/>
         <xsd:element ref="field2"/>
         <xsd:element ref="field3"/>
      </xsd:all>
   </xsd:complexType>
</xsd:schema>
```

```
[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://example.org/")]
 [System.Xml.Serialization.XmlRootAttribute("complexInstance", Namespace="http://example.org/",
IsNullable=false)]

public class MyComplexType {

   public string field1;

   public string field2;

   public string field3;
}
```

Listing 4: C# class generation with XSD-Tool – a simple example [28].

5.3 Generation of the EMF Data Model

Since the specification differencing engine is based on the EMF, it is also necessary to create an EMF data model out of our XML files. For this purpose the XML schema model importer included in the EMF must be used to convert the XSD meta-model to an appropriate EMF ecore meta-model. As for complex meta-models this conversion procedure is a very time-consuming process. It is performed once during development and the resulting EMF ecore meta-model is

stored for further use. Thus, in case the XSD meta-model changes the EMF ecore meta-model must be regenerated.

After the EMF ecore meta-model has been generated the *ResourceSet* class can be used to load content of XML files into the EMF data model and to create the according *EObject* classes containing the model data. Once the XML data is imported EMF can operate with the model data. In our implementation this is used for the generation of comparison models and thus to detect changes between similar data models.

5.4 C# Data Model Access

According to the requirements for the optimized development process listed in Section 4.2 the requirements and specification artifacts shall be visualized in tabular form. As the C# data model is used for the operation with this data, a mechanism needs to be implemented that preprocesses the data object tree of the data model and converts it to an according table representation.

The data model and its preprocessing mechanism is implemented in a separate Data Model Library (DML), which is described in this section. This DML also contains the C# classes generated by the XSD tool and uses the *XMLSerializer* library (XMLSL) to read XML documents and create the according C# object model.

The XMLSL basically consists of two important classes, which aim to read and deserialize XML content. The so created data object structure can then be accessed by the DLM for further processing.

`XMLSerializer`

> This class is responsible to load XML content by using the Microsoft .NET serializer classes and create an according object structure out of the XML data.

`Settings`

> This class contains the definitions of the standard namespace used in the requirements and specification artifact XML files.

The DML implements an interface that is described below and allows accessing the preprocessed data.

`List<DataRecord> GetData(string filePath)`

Uses the *XMLSerializer* library to read requirements and specification artifacts stored in XML documents, preprocesses the data and returns a list of *DataRecord* objects, where a *DataRecord* represents a row in the data table.

`void StoreData(string filepath)`

Uses the *XMLSerializer* library to store the loaded data object model in an appropriate XML format.

`DataRecord GetDataRootElement()`

Returns the *DataRecord* object containing the highest hierarchical data item, the root node of the object tree.

`string GetXmlStandardNamespace()`

Returns the default XML namespace set up in the *Settings* class of the DML. All namespaces used in requirements and specification artifact XML files must be registered in the DML.

`List<InsertMenuItem> GetListOfPossibleSubitems(object item)`

Returns a list of possible kinds of subitems that can be inserted below the given data item.

`bool IsDisplayedAsRow(PropertyInfo propertyInfo)`

Indicates if the given property contained within the data object model is displayed as row or column in the visualization of the data table. Properties that can contain only a single element, which is representable as text single cell, are displayed as row, whereas elements of arrays and objects that are not representable in a single cell are depicted as row.

`string GetItemRowTypeString(PropertyInfo parentItemPropertyInfo, object item)`

Returns a string indicating the type of the object represented by a certain property of the data object model.

After preprocessing of the deserialized data a single row of the created data table is represented by a *DataRecord* object. Such an object basically contains the information listed below.

`int Level`

The hierarchical level of the data item within the data model object tree.

`object Content`

The actual content of the represented data item.

`object ParentItem`

The parent data item in the hierarchical structure.

`object DirectParentItem`

The array or container of the parent item that contains the representing data item.

`string ContentPropertyName`

The name of the property that contains the representing data item.

`string Type`

The type of the representing data item as string.

The *DataProcessor* class included in the DML is basically responsible for data preprocessing. Its main task is to grab XML data by using the *XMLSerializer* class and create a list of *DataRecord* objects out of the data object tree. For this purpose the data needs to be interpreted and the implemented logic must decide how the data is represented in the data table of the user interface. After the preprocessing process a list of *DataRecord* objects will be returned. Since each *DataRecord* object represents a single row of the data table displayed in the user interface, this data representation facilitates the visualization of the requirements and specification artifacts in the user interface.

The following criteria are used to decide how data is represented in the data table:

- Properties with an elementary data type (int, bool, double, float, string, char) are displayed as column.
- Properties with data type object and without an attribute definition of type *XMLElementAttribute* are also displayed as column.
- Enums are displayed as column.
- Enum arrays are displayed as column.
- In all other cases data is represented as row in the data table.

5.5 Differencing Engine

The Requirements and Specification Differencing Engine (RSDE) takes two XML files containing any requirements and specification artifacts and a pre-generated ecore model as input. Then, it calculates the comparison model and produces an output containing all differences between the two models.

The implementation of the RSDE is based on the EMF Compare framework [14], which is also described in Section 3.2.2. This is a highly customizable framework that facilitates operating with model data. The RSDE is based on the *EMF Compare Standalone example implementation*[7], which has been modified and extended to calculate comparison models of requirements and specification artifacts stored in XML documents. The basic structure of the implementation is described in this section.

The RSDE basically consists of six classes:

`DiffLauncher`

This class contains the entry point of the RSDE. It checks the launch arguments, creates an instance of the *DiffEngine* class and is responsible to start the calculation of the comparison model.

`DiffEngine`

This class contains the implementation of the EMF Compare based differencing mechanism.

`EMFCompareXmlPrinter`

This class contains functions to generate an XML document representing the differences of the comparison model.

`AttributeValue`

This class represents an attribute value pair of an ecore model and is needed for working with the content of ecore objects.

[7] https://github.com/cbrun/emf-compare/tree/master/plugins/org.eclipse.emf.compare.examples.standalone

`Settings`

This class contains static settings for the RSDE.

`Utils`

This class contains helper functions for several purposes.

When the RSDE is started, first all resources such as ecore models and XML documents, that are to be compared, are loaded. Then, an EMF comparator is instantiated and configured, which is a processor that calculates the comparison model. The configuration of the comparator is a very important step and defines the way how models are compared to each other. A detailed description of the configuration of the comparison mechanism can be found in the next section. Finally, the comparison model is calculated and the differences stored in an XML document for further processing. The visualization of the output needs to be done by a separate application.

5.5.1 Configuration of the EMF Comparator

The EMF comparator contains the comparison algorithms and must be configured by the developer. Thus, the way how models are compared to each other must be defined by the developer. For this purpose either a predefined mechanism can be used or if none of them is suitable, a so-called custom matcher needs to be implemented.

In our case we would like to compare requirements and specification artifacts to each other. Since most of them have an identifier, that is unique at least within a single module, a custom matcher has been implemented that uses these identifiers to find matching artifacts. This way of work increases the performance of the comparison algorithm tremendously. A matcher is a part of the differencing engine and is responsible to find matching artifacts that can then be compared to each other. Listing 5 shows the implementation of the used matching function. This piece of code defines a function that tries to get and return the ID of the given ecore object. In the case no ID attribute can be found, null is returned and the fall back matcher will be triggered.

As fall back matcher the default matching engine of the EMF Compare framework is used. As stated in [11] and described in more detail in Section 3.2 of this work the default matching engine uses a similarity-based matching approach. This has also some disadvantages, which are described in the next section. Listing 6 shows the source code necessary for the configuration of the EMF Compare matchers. Within the *configEMFCompare* function a fall back matcher and a custom ID matcher is instantiated and the matcher engines are configured. The fall back matcher uses the ID match function defined in Listing 5 to identify appropriate matches. Finally

the *configEMFCompare* function returns an *EMFCompare* instance that can then be used for the calculating the comparison model with the selected configuration.

```
Function<EObject, String> idFunction = new Function<EObject, String>()
  {
    public String apply(EObject input)
    {
      String idValue = Utils.getIDAttributeValue(input);
      if (idValue != "")
      {
        return idValue;
      }
      else
      {
        return null;
      }
    }
  };
```

Listing 5: Implementation of the ID match function.

```
private EMFCompare configEMFCompare()
  {
    // instantiate the fallback for the case that an ID is not available
    IEObjectMatcher fallBackMatcher =
      DefaultMatchEngine.createDefaultEObjectMatcher(UseIdentifiers.WHEN_AVAILABLE);

    // instantiate an ID matcher for the case that an ID is available
    IEObjectMatcher customIDMatcher = new IdentifierEObjectMatcher(fallBackMatcher, idFunction);

    // configure the matcher engines
    IComparisonFactory comparisonFactory = new DefaultComparisonFactory(
      new DefaultEqualityHelperFactory());
    IMatchEngine.Factory matchEngineFactory = new MatchEngineFactoryImpl(
      customIDMatcher, comparisonFactory);
    matchEngineFactory.setRanking(20);
    IMatchEngine.Factory.Registry matchEngineRegistry = new MatchEngineFactoryRegistryImpl();
    matchEngineRegistry.add(matchEngineFactory);
    EMFCompare.Builder emfCompareBuilder =
      EMFCompare.builder().setMatchEngineFactoryRegistry(matchEngineRegistry);
    EMFCompare comparator = emfCompareBuilder.build();

    return comparator;
  }
```

Listing 6: Configuration of the EMF Compare matcher.

5.5.2 Limitations of the EMF Compare Framework

The EMF Compare engine has also some limitations that have been experienced during the implementation of this work.

The experienced limitation concerns the performance of the matching engine. If an artifact does not contain a unique identifier, the fall back matcher needs to be used. In this case basically an EMF default matching engine is used, which follows a similarity-based matching approach. When differencing large models with a huge number of differences by using such an approach, performance problems have been experienced at runtime. Performance and memory problems have also been noticed when two large models are compared that have exactly the same content, but with attributes defined in different orders.

Such kinds of problems can be avoided by making sure that each artifact has its own unique identifier. Thus, the fall back matcher does not need to be used and the differencing mechanism works efficiently.

5.6 User Interface of the Requirements and Specification Management System

The RSMS developed in the scope of this work allows the user to load several projects and to operate with requirements and specification artifacts. Add, delete and modify operations can be applied to design artifacts and comparison models can be visualized.

Figure 19 gives an overview of the basic structure of the RSMS. The implementation is basically structured into the packages described below.

`Data`

This package basically consists of two classes: the *Project* class, which represents a loaded project with all its properties and the *ProjectManager* class containing all functions to operate with projects. In order to be able to store project settings in a file and to recover them when restarting the application, the *Project* class must implement the *ISerializable* interface.

`DiffMerge`

This package contains mechanisms to control the differencing engine running in an external process and to manage its output. The differencing engine is described in the section below. Additionally it implements a parser that reads the output of the differencing engine and a differences processor that interprets and optimizes differences.

`ExternalProcesses`

This package contains classes that can be used to create, start and control external processes. Special processes such as that one for the differencing engine are derived from this class.

`UserControls`

All user controls such as the requirements and specification editor are located in this package.

`UserInterface`

This package contains the implementation (classes and forms) of the graphical user interface.

`Util package`

This package contains classes that offer small helper functions for several purposes.

5.6.1 Requirements and Specification Editor

This section describes the way of work of the requirements and specification editor. In order to encapsulate the functionality of this editor and to allow multiple instances of the editor it is implemented as user control element. It builds on the list of *DataRecord* objects produced by the DML. Since the data has been preprocessed in a proper way, each *DataRecord* corresponds to a single row in the data table visualized in the editor.

The main task here is to interpret the content of each attribute and visualize it in a proper way.

The Requirements and Specification Editor (RSE) uses the DML to get a list of preprocessed *DataRecord* objects. Thus, the RSE must only run through the list and add the content of each *DataRecord* object to the data table. During this process it is necessary to interpret this content and to display each property in an appropriate way. In the data table of the user interface data can be displayed by using one of the following control elements:

- **Text field:** For properties containing general data.
- **Combo box:** For properties with type of enum
- **Combo box with check boxes:** For properties with type of enum array.

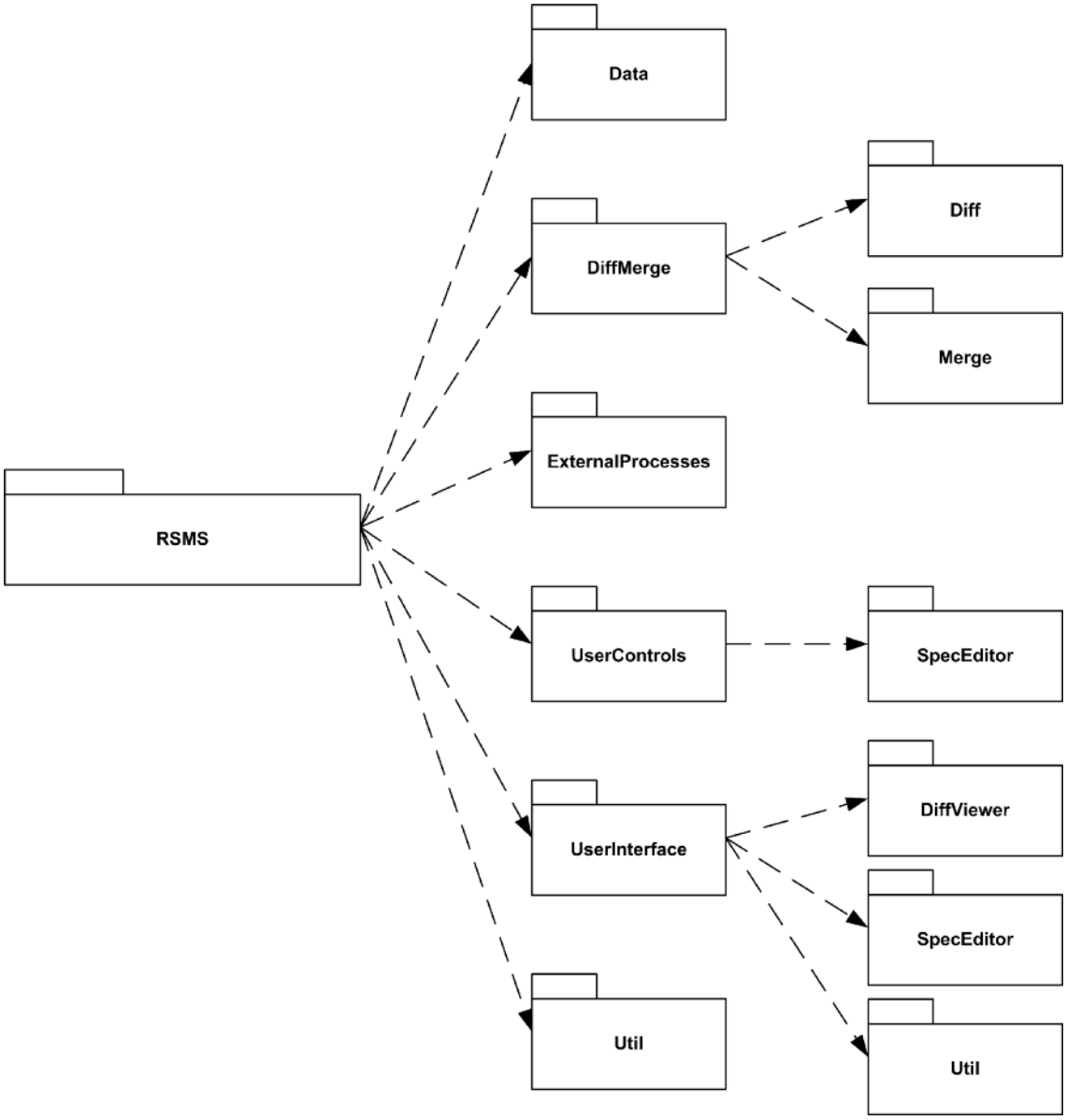

Figure 19: Overview of the RSMS structure.

5.6.2 Visualization of Comparison Models

In order to create and visualize a comparison model the external differencing engine is started by using the *DiffManager* class. This gets as input the XML files that are to be compared and the ecore model, which has been generated before (see Section 5.3). Then, the differencing engine analyzes the differences of the two XML requirements and specification models and calculates an according comparison model, which is again stored in an XML document. Listing 7 shows such an example output produced by the differencing engine.

```
<Diffs>
  <Diff>
    <Operation>ADD</Operation>
    <Kind>ReferenceChange</Kind>
    <IDAttribute>ID</IDAttribute>
    <IDValue>25</IDValue>
  </Diff>
  <Diff>
    <Operation>DELETE</Operation>
    <Kind>ReferenceChange</Kind>
    <IDAttribute>ID</IDAttribute>
    <IDValue>25</IDValue>
  </Diff>
  <Diff>
    <Operation>CHANGE</Operation>
    <Kind>AttributeChange</Kind>
    <ChangedAttribute>status</ChangedAttribute>
    <ChangedValue>Rejected</ChangedValue>
    <IDAttribute>ID</IDAttribute>
    <IDValue>1493</IDValue>
  </Diff>
  <Diff>
    <Operation>MOVE</Operation>
    <Kind>AttributeChange</Kind>
    <IDAttribute>ID</IDAttribute>
    <IDValue>1493</IDValue>
  </Diff>
</Diffs>
```

Listing 7: Example output of the differencing engine.

After that the output of the differencing engine is parsed by using the *DiffFileParser* class, which stores each detected difference in a *Difference* object. This object contains the following information:

`string Kind`

Kind of difference.

`string Operation`

Kind of operation applied to a data object.

`string OperationText`

Description text for the operation applied to a data object.

`string IdAttribute`

Name of the attribute that contains the unique identifier.

`string IdValue`

The unique identifier of the data object that contains the difference.

In the next step the *DifferenceProcessor* class is used to optimize and filter the list of differences created by the parser. This is necessary due to some limitations of the EMF Compare engine discussed in section 5.5. The output of the EMF Compare engine contains the name of the detected operation, the kind of operation, the name of the value, where the ID is stored in and the ID itself. Based on this information the following optimizations are applied. For each optimization measure it is described how a certain result must be interpreted. The optimized list of differences is then used to visualize them in the graphical user interface.

Optimization 1:

Detection: An *ADD* and *DELETE* operation has been detected on the same ID and the kind of operation is *AttributeChange*.

Interpretation: This means that an attribute of a subordinated data object changed. Thus, these originally two *Difference* objects result in a single *Difference* object with the operation text "Subcontent Modified".

Optimization 2:

Detection: An *ADD* and *DELETE* operation on the same ID has been detected and the kind of operation is *ReferenceChange*.

Interpretation: This means that an attribute of the according data object changed. These originally two *Difference* objects result in a single *Difference* object with the operation text "Modified".

Clicking on a difference in the differences list of the differences view in the user interface highlights and shows the according difference in the RSE. Figure 21 in Chapter 6 shows a screenshot of this use case.

Chapter 6

Evaluation

This chapter evaluates the optimizations designed and implemented during this work. For this reason the evaluation is divided into three parts. Section 6.1 evaluates the performance of a DOORS export needed for several operations in the former development process and compares this to the performance of the optimized development and security evaluation process. Additionally, Section 6.2 shows the usage of the requirements and specification differencing tool by calculating the comparison model of two similar example requirements models. Finally, Section 6.3 evaluates the results of this work by using the goals question metric method.

6.1 Export-Performance for Reuse of Data - a Comparison

The first evaluation puts the focus on the export performance of the requirements management system DOORS, which has been used in the former development process. All kinds of requirements and specification artifacts have been stored in this system. Thus, for the further automated processing of any artifacts, the data was needed to be exported into a CSV file. This is the standard export format of DOORS and it is a file format where a data record consists of a single line in a text file and values of different columns are separated by a semicolon.

The export process was necessary especially for the following purposes:

- Generation of the documentation.
- Generation of firmware interfaces (C header files).
- Generation of hardware interfaces (Verilog files).

The performance of this export process is quite poor and thus it is very time-consuming. These problems and their consequences, especially the negative impact on the agile development, are also discussed in Chapter 2. For this evaluation, we measured the export performance of

DOORS modules with different amounts of data. In order to establish a remote connection to the DOORS database a local DOORS Client has been used.

Table 8 lists the export performance dependent on the amount of data exported from DOORS. The results are also illustrated in Figure 20. A closer view on the result shows that the time needed for the data export increases linearly with the amount of data transferred from the DOORS database to the local computer.

Furthermore, it has been evaluated that a project with a size of about 35 Megabyte of specification data (which corresponds to a project of an average size) needs approximately 9 hours to be exported.

This is one aspect that has been optimized within the scope of this work. Since in the optimized, unified development process the specification is stored directly in an XML document (see also Section 4.4.1), an export process is no longer necessary. Instead, the specification is stored in a text-based format and it is structured in a way that allows accessing the information directly without the need of applying further time-consuming conversion processes.

The result is an enormous gain of performance and thus, short round trip times, which are suitable for agile development.

Transferred data in kB	DOORS Export Performance in min
11	1.11
110	4.71
669	10.47
980	15.97

Table 8: DOORS export performance.

6.2 Visualization of Comparison Models

This section describes the use of the RSMS described in Section 4.4.1 for visualizing a comparison model. This is explained by using a simple example. For this demonstration two similar XML files with specification artifacts have been created, which differ only in some

details. These differences are to be analyzed and visualized in the developed requirements management system.

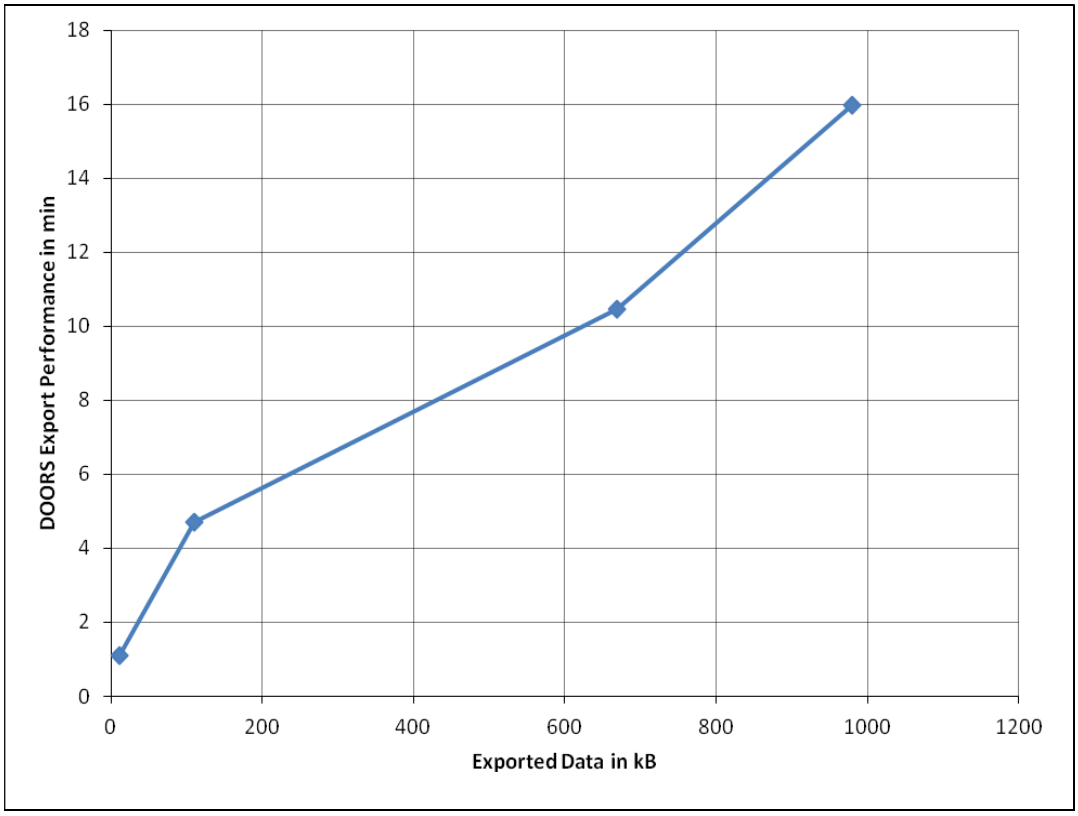

Figure 20: DOORS export performance as function of the amount of exported data.

The XML document containing the sample requirements is shown in Listing 8. This file contains a scope *TestScope*, 3 nested groups and 4 sample requirements.

The modified XML document is shown in Listing 9. All modifications are highlighted in red and listed below:

- Text of requirement with ID = 1 changed.
- Status attribute of requirement with ID = 3 changed.
- Text of requirement with ID = 3 changed.
- The order of requirement with ID = 3 and ID = 4 changed.

Then, the differencing engine described in Section 5.5 is used to generate a comparison model of the specification presented above.

This application can either be started directly out of the requirements management system or in a console by using the following command:

java –jar diffengine.jar [param1] [param2] [param3]

param1	Path to the ecore model.
param2	Path to the base revision of the XML document containing the design artifacts.
param3	Path to the XML document containing the modified design artifacts.

```
<Specs>
  <Scope Name="TestScope" Type="Mixed">
    <Source Name="Main">
      <Group ID="10" Name="">
        <Heading>Example Requirements</Heading>
        <RS:RS />
        <Group ID="11" Name="Ext_Del">
          <Heading>My First Group</Heading>
          <RS:RS />
          <Group ID="12" Name="Ext_Del_General">
            <Heading>General</Heading>
            <RS:RS>
              <RS:Req ID="1" Status="Accepted" Priority="must have" Satisfied="N/A" Qualified="not">
                <RS:Text>This is the text of my first requirement.</RS:Text>
              </RS:Req>
              <RS:Req ID="2" Status="Rejected" Priority="must have" Satisfied="yes" Qualified="TBD">
                <RS:Text>This is the text of my second requirement.</RS:Text>
              </RS:Req>
              <RS:Req ID="3" Status="Modifying" Priority="nice to have" Satisfied="yes" Qualified="yes">
                <RS:Text>This is the text of my third requirement.</RS:Text>
              </RS:Req>
              <RS:Req ID="4" Status="Accepted" Priority="must have" Satisfied="yes" Qualified="yes">
                <RS:Text>This is the text of my fourth requirement.</RS:Text>
              </RS:Req>
            </RS:RS>
          </Group>
        </Group>
      </Group>
    </Source>
  </Scope>
</Specs>
```

Listing 8: Content of the XML file containing the requirements (base revision).

```
<Specs>
  <Scope Name="TestScope" Type="Mixed">
    <Source Name="Main">
      <Group ID="10" Name="">
        <Heading>Example Requirements</Heading>
        <RS:RS />
        <Group ID="11" Name="Ext_Del">
          <Heading>My First Group</Heading>
          <RS:RS />
          <Group ID="12" Name="Ext_Del_General">
            <Heading>General</Heading>
            <RS:RS>
              <RS:Req ID="1" Status="Accepted" Priority="must have" Satisfied="N/A" Qualified="not">
                <RS:Text>This is the modified text of my first requirement.</RS:Text>
              </RS:Req>
              <RS:Req ID="2" Status="Rejected" Priority="must have" Satisfied="yes" Qualified="TBD">
                <RS:Text>This is the text of my second requirement.</RS:Text>
              </RS:Req>
              <RS:Req ID="4" Status="Accepted" Priority="must have" Satisfied="yes" Qualified="yes">
                <RS:Text>This is the text of my fourth requirement.</RS:Text>
              </RS:Req>
              <RS:Req ID="3" Status="Rejected" Priority="nice to have" Satisfied="yes" Qualified="yes">
                <RS:Text>This is the text of my third requirement with a modified attribute.</RS:Text>
              </RS:Req>
            </RS:RS>
          </Group>
        </Group>
      </Group>
    </Source>
  </Scope>
</Specs>
```

Listing 9: Content of the XML file containing the requirements (modified).

Based on the XML documents presented above, this engine generates the XML output representing the calculated comparison model (see Listing 10).

The user interface parses this comparison model, applies optimizations, which are discussed in Section 5.6.2, and shows them finally as follows:

ID 1: MODIFIED

ID 3: MODIFIED

ID 3: MOVED

Additionally, the user can select one of these changes and gets a comparison view of the affected data record in the compared models. Figure 21 shows the visualization of the comparison model.

```
<Diffs>
  <Diff>
    <Operation>ADD</Operation>
    <Kind>ReferenceChange</Kind>
    <IDAttribute>ID</IDAttribute>
    <IDValue>1</IDValue>
  </Diff>
  <Diff>
    <Operation>DELETE</Operation>
    <Kind>ReferenceChange</Kind>
    <IDAttribute>ID</IDAttribute>
    <IDValue>1</IDValue>
  </Diff>
  <Diff>
    <Operation>ADD</Operation>
    <Kind>ReferenceChange</Kind>
    <IDAttribute>ID</IDAttribute>
    <IDValue>3</IDValue>
  </Diff>
  <Diff>
    <Operation>DELETE</Operation>
    <Kind>ReferenceChange</Kind>
    <IDAttribute>ID</IDAttribute>
    <IDValue>3</IDValue>
  </Diff>
  <Diff>
    <Operation>CHANGE</Operation>
    <Kind>AttributeChange</Kind>
    <ChangedAttribute>status</ChangedAttribute>
    <ChangedValue>Rejected</ChangedValue>
    <IDAttribute>ID</IDAttribute>
    <IDValue>3</IDValue>
  </Diff>
  <Diff>
    <Operation>MOVE</Operation>
    <Kind>AttributeChange</Kind>
    <IDAttribute>ID</IDAttribute>
    <IDValue>3</IDValue>
  </Diff>
</Diffs>
```

Listing 10: Output of the differencing engine.

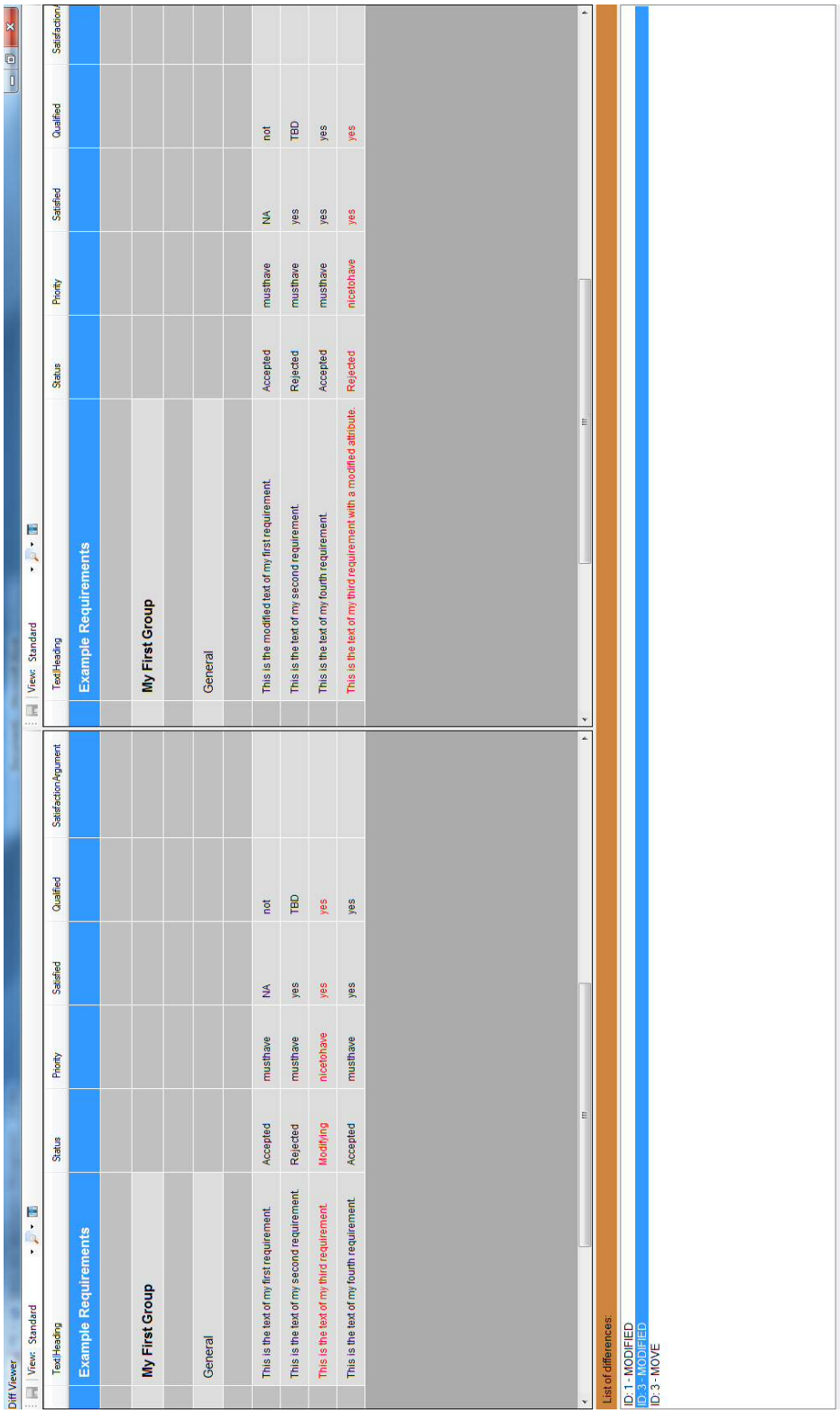

Figure 21: Visualization of the comparison model in the RSMS.

6.3 Evaluation of the Results with Goals Question Metric

Goal Question Metric (GQM) [30] is a systematic approach for creating a quality model. In this section we used such an approach to measure the quality of the results of this work. This analysis considers not only the implemented parts, but also the overall design described in Chapter 4.

In the first step we defined measurement goals. Then, we elaborated questions that define the goals in more detail. Abstraction sheets (see Table 9, Table 10, Table 11 and Table 12) have been used to collect important data related to the goals. In the next step metrics have been derived from the questions to measure the goals. The collected data has been evaluated (see Table 14, Table 15, Table 16, Table 17 and Table 18) and a GQM plan has been derived, which is shown in Figure 22.

GQM Abstraction Sheet			
Goal:	G 1.1.1: Shorten the round trip time for FW interface generation.		
Point of view:	Developer	**Environment:**	Development process for IC development
Quality focus		Variation factors	
1. Number of steps needed until a changed interface definition is available in the source code. 2. Number of applications needed in parallel to change an interface definition.		1. Number and complexity of the applications to be used. 2. Amount of exported data.	
Baseline hypothesis		Environment impact on baseline hypothesis	
1. At least 2 applications (DOORS, Development Environment) are necessary to generate C header files. 2. At least 1 export and 2 conversion processes are necessary to generate a C header file. 3. DOORS data export processes are very time-consuming.		1. Number of applications that need to be used in parallel influences baseline hypothesis 1. 2. Amount of interface data that need to be exported influences baseline hypothesis 3. 3. High number of export and conversion processes influences baseline hypothesis 2.	

Table 9: GQM abstraction sheet: Round trip time for SW/FW development.

GQM Abstraction Sheet			
Goal:	G 1.4.1: Reuse and referencing of design artifacts in the documentation.		
Point of view:	Developer	**Environment:**	Development process for IC development
Quality focus		Variation factors	
1. Redundancies of design data in the development process and in the documentation. 2. Inconsistencies in the documentation. 3. High effort for product maintenance.		1. Need of copy and paste operations for referencing design artifacts in the documentation. 2. Missing automation mechanism for referencing design artifacts in the documentation.	
Baseline hypothesis		Environment impact on baseline hypothesis	
1. Content of design artifacts redundantly available in the documentation. 2. Inconsistent documentation.		1. Referencing of design artifacts by using copy and paste operations influences baseline hypothesis 1 and 2. 2. Manual changes of the content of design artifacts in the documentation influences baseline hypothesis 2.	

Table 10: GQM abstraction sheet: Resuse and referencing of design artifacts in the documentation.

GQM Abstraction Sheet			
Goal:	G 1.2.1: Support multi-user access for the requirements editor.		
Point of view:	Developer	**Environment:**	Development process for IC development
Quality focus		Variation factors	
1. Number of users that can make changes concurrently in a module.		1. Functionality and mechanisms for teamwork support in the RMS.	
Baseline hypothesis		Environment impact on baseline hypothesis	
1. Only 1 person can work on a module.		1. Functionality and mechanisms of the RMS influences baseline hypothesis 1.	

Table 11: GQM abstraction sheet: Multi-User Access.

GQM Abstraction Sheet			
Goal:	G 1.3.1: Support custom meta-model for design artifacts.		
Point of view:	Developer	**Environment:**	Development process for IC development
Quality focus		Variation factors	
1. Quality of the stored design data. 2. Support for creating new design data.		1. Data validation	
Baseline hypothesis		Environment impact on baseline hypothesis	
1. Erroneous and incomplete design data. 2. Additional effort of maintenance for erroneous and incomplete design data.		1. Data validation when inserting/creating new design data influences baseline hypothesis 1 and 2.	

Table 12: GQM abstraction sheet: Custom meta-model support.

GQM Abstraction Sheet			
Goal:	G 2.1.1: Unified process for development and security evaluation.		
Point of view:	Developer	**Environment:**	Development process for IC development
Quality focus		Variation factors	
1. Complete linking between design artifacts and artifacts for security evaluation.		1. Links between design artifacts and artifacts of the security evaluation process. 2. Structure of the process for development and security evaluation.	
Baseline hypothesis		Environment impact on baseline hypothesis	
1. Separated documentation for design and security evaluation. 2. No tracing between design artifacts and artifacts of the security evaluation process possible.		1. Missing links between design artifacts and artifacts of the security evaluation process influence baseline hypothesis 2. 2. Two separated processes for development and security evaluation influence baseline hypothesis 1 and 2.	

Table 13: GQM abstraction sheet: Unified process for development and security evaluation.

Goal:	**G 1.1.1**
Description:	Shorten the round trip time for FW interface generation
Question Q1:	How much applications are needed in parallel to modify an interface definition?
Metric M1:	Find the number of applications necessary to modify an interface definition in the source code.
Answer A1:	Only one application is needed for modifying an interface definition. Rationale: Interface definitions can directly be modified in the source code. The synchronization of the source code with the RMS must not be done at development time. Thus, only a development environment for editing the source code is necessary.
Question Q2:	How much data export and conversion processes need to be executed to synchronize the RMS with the source code?
Metric M2:	Find the number of export and conversion processes necessary for synchronizing the source code and the data in the RMS.
Answer A2:	No data export and one conversion process is needed for the synchronization. Rationale: Since specification is stored in XML documents, the specification can be directly processed without the need of an export process. A conversion process is necessary to generate source code out of the XML specification.

Table 14: Analysis of goal G 1.1.1.

Goal:	**G 1.2.1**
Description:	Support multi-user access for the requirements editor.
Question Q 3:	How much team members can work on a module concurrently?
Metric M 3:	Find the number of users that can work concurrently on a module in the RMS.
Answer A 3:	An arbitrary number of users can work concurrently an a module. Rationale: Each user works on a local copy of the specification. When a user commits the changes, a merge engine is used to merge the user's changes into the head revision of the modified module.

Table 15: Analysis of goal G 1.2.1.

Goal:	**G 1.3.1**
Description:	Support custom meta-models for design artifacts.
Question Q4:	Is the design data validated after the insertion process?
Metric M4:	Find out if new inserted data is validated before storing them in the RMS.
Answer A4:	No, data is currently not automatically validated before storing them in the RMS. Rationale: This is a feature that must be implemented within the scope of a further work. Nevertheless, specification can be easily validated by using an XSD check. This return true if the specification data is valid and false otherwise. A pointer to the erroneous data record cannot be derived from the output of an XSD check.

Table 16: Analysis of goal G 1.3.1.

Goal:	**G 1.4.1**
Description:	Reuse and referencing of design artifacts in the documentation.
Question Q5:	Can design artifacts be inserted into the documentation without the use of copy and paste operations?
Metric M5:	Find out if design artifacts can be referenced in documents without copy and paste operations.
Answer A5:	Yes. Rationale: Design artifacts can be easily referenced in the documentation by using special LaTeX commands. A documentation engine resolved the references and inserts the according content automatically.
Question Q6:	The content of a design artifact has been changed. On how much locations must information be adapted in order to avoid inconsistencies?
Metric M6:	Find the number of locations, where data must be changed if the content of a design artifact changes.
Answer A6:	1. Rationale: Since a *single-point-of-source* approach has been followed, the content of specification artifacts must only be changed in the specification repository. All references in the documentation art updated automatically.

Table 17: Analysis of goal G 1.4.1.

Goal:	**G 2.1.1**
Description:	Unified process for development and security evaluation
Question Q7:	How much links contains a module of average size?
Metric M7:	Find the number of links a module of average size contains.
Answer A7:	1700 links. Rationale: A module on system level for a security IC with about 1400 requirements contains approximately 1700 links (inbound and outbound links).

Table 18: Analysis of goal G 2.1.1.

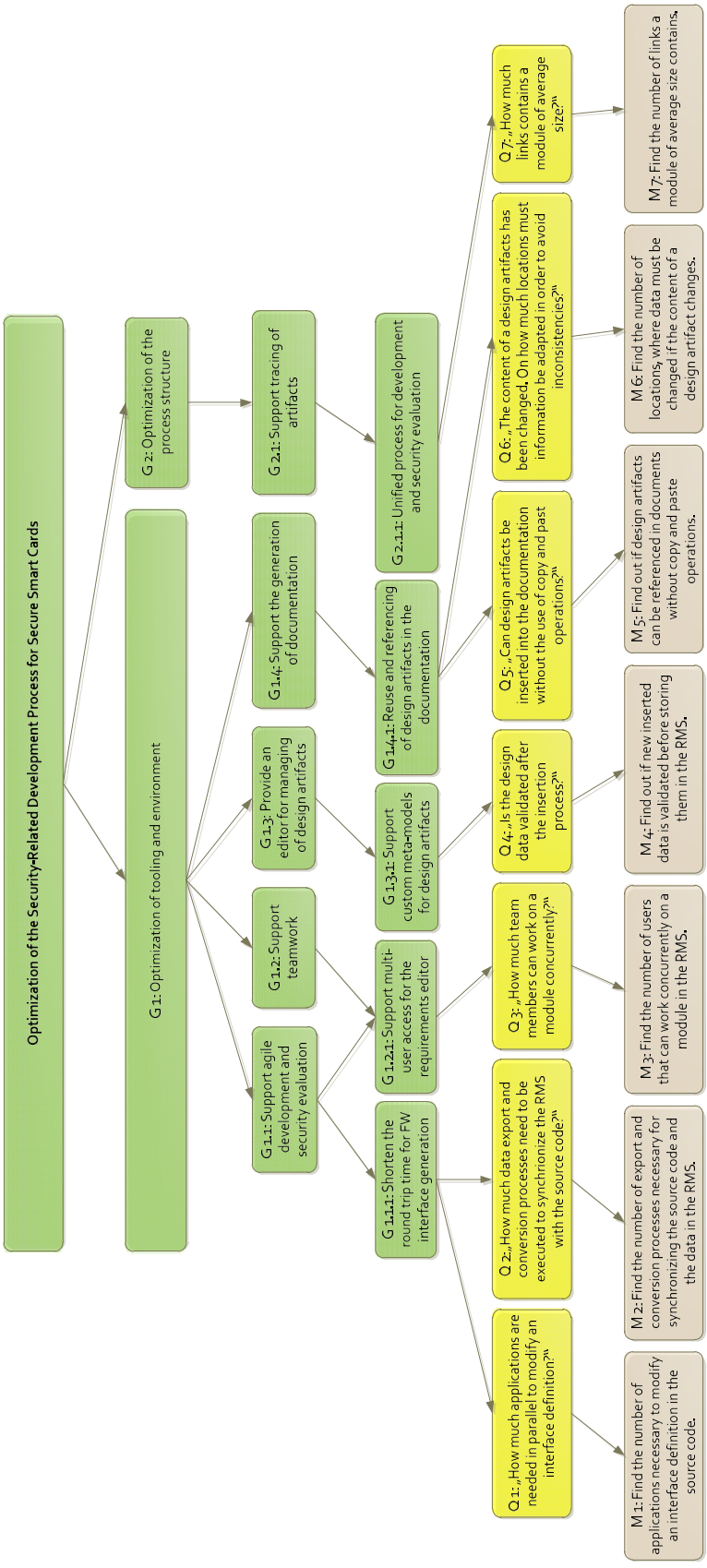

Figure 22: GQM plan.

Chapter 7

Results and Outlook

In this work the development and security evaluation process for secure smart cards has been optimized with regard to the used tooling environment. The goal was to optimize the development process and to implement tools that support developers in doing their daily work.

Since this work has been created in cooperation with NXP Semiconductors Austria GmbH, it started with an analysis of the current situation of the development process in this company. With this analysis, we were able to identify the problems of the existing process. In the next step we defined a set of goals for the optimized development process. One of the goals was to combine the development and security evaluation process and to find an appropriate framework supporting the developers in creating the documentation. Thus, we did a research on existing technologies for producing engineering documents. We analyzed and compared technologies that allow referencing and tracing design artifacts in the documentation and we found out that the ADE development framework suits best for our purposes. This documentation framework had been developed in the scope of a previous work and was now reused in the optimized process. Additionally, we needed to find a suitable requirements management system. This should allow managing different types of design artifacts such as requirements, software interface specification and test specification. Artifacts needed for the security evaluation process such as security problem definition or security objectives should also be managed by using this application. Since we did not find an appropriate existing system, that satisfies all our requirements, we decided to implement a prototype of such a solution by ourselves. Our approach allows the user to define meta-models for different specification types, which makes the system flexible and highly customizable. Furthermore, a differencing engine has been implemented. This implementation is based on the EMF Compare framework and facilitates the user to calculate comparison models showing the differences of two models containing design artifacts. A graphical user interface visualizes the detected differences and makes them accessible to the user.

Our implementations that have been created within the scope of this work have also some limitations, which are discussed here. One of them is the missing XSD check of the input in the

requirements and specification editor. A XSD check is an XML schema validation that determines if the given XML file follows the rules defined in the XSD. In case of a negative XSD check the data record that causes the error cannot be identified. Thus, the results of this check can also not be visualized in the user interface.

Since the effort for implementing the entire process would go beyond the scope of this work, only a part of the design presented in this work has been implemented.

Chapter 4 describes a differencing and merge engine needed for the implementation of the synchronization layer shown in Figure 9. Within the scope of this work the differencing engine has been implemented. It is planned to implement the merge engine in the scope of a further work. The according process has already been considered in the design described in Chapter 4.. The EMF Compare framework, which is currently used for calculating the comparison models, offers also comfortable opportunities to merge design artifacts of different sources. Thus, in the next step the implementation of the differencing engine must be extended. Furthermore it would be necessary to create an interactive connection between the graphical user interface and the according merge engine, which allows the user to select the design artifacts that are finally to be inserted into the merged model.

Furthermore, according to the description in Section 4.5, the two separated processes for development and security evaluation have been combined to a single, unified process. The design artifacts stored in XML documents contain now links, which are the basis for the traceability of design artifacts and for impact analyses. The documentation engine uses also this information to display linked artifacts in the design and security evaluation documentation. A user interface to create and modify such links needs still to be implemented and is also within the scope of a further work.

Parts of the results of this work have also been published in the paper *Supporting Evolving Security Models for an Agile Security Evaluation* [31].

Chapter 8

Bibliography

[1] G. Matthew Ezovski and Steve E. Watkins. The Electronic Passport and the Future of Government-Issued RFID-Based Identification. In *RFID, 2007. IEEE International Conference on*, pages 15–22, March 2007.

[2] Klaus Finkenzeller. RFID handbook: fundamentals and applications in contactless smart cards and identification. *Hardcover*, 2003.

[3] Capgemini and RBS. World Payments Report 2013. Capgemini, 2013.

[4] Damien Sauveron and Pierre Dusart. Which trust can be expected of the Common Criteria certification at end-user level? In *Future Generation Communication and Networking (FGCN 2007)*, volume 2, pages 423–428, Dec 2007.

[5] ISO/IEC_JTC1/SC27. Information technology-Security techniques-Evaluation criteria for ITsecurity. ISO/IEC 15408:2006 (Common Criteria v3.1), 2006.

[6] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid. Dynamic Software Product Lines. *Computer*, 41(4):93–95, April 2008.

[7] Linda Northrop. Software Product Lines. http://http://splc.sei.cmu.edu/library/assets/Philips.04.12.05.pdf, 2001.

[8] Zsuzsa Varvasovszky and Ruairí Brugha. A stakeholder analysis. *Health Policy and Planning*, 15(3):338–345, 2000.

[9] Debra S. Herrmann. Using the Common Criteria for IT security evaluation. CRC Press, 2002.

[10] Security IC Platform Protection Profile, registered and certified by Bundesamt fuer Sicherheit in der Informationstechnologie (BSI) under the reference BSI-CC-PP-0084-2014, Rev. 1.0, 2014

[11] Dimitrios S Kolovos, et. al. Different models for model matching: An analysis of approaches to support model differencing. In *Comparison and Versioning of Software Models, 2009. CVSM'09. ICSE Workshop,* pages 1-6. IEEE, 2009.

[12] Cédric Brun and Alfonso Pierantonio. Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional*, 9(2):29-34, 2008.

[13] Tancred Lindholm. A 3-way Merging Algorithm for Synchronizing Ordered Trees–the 3DM merging and differencing tool for XML. *Master thesis, Helsiniki University Of Technology,* 2001.

[14] Eclipse Foundation. EMF Compare. http://www.eclipse.org/emf/compare, 2008.

[15] Jeremy Dick. Design traceability. *Software, IEEE,* 22(6):14-16, 2005.

[16] Michael Priestley. DITA XML: A reuse by reference architecture for technical documentation. In *Proceedings of the 19th annual international conference on Computer documentation*, pages 152-156. ACM, 2001.

[17] Kristen James Eberlein, et al., Darwin Information Typing Architecture (DITA) Version 1.2. *Organization for the Advancement of Structured Information Standards, OASIS*, 2010.

[18] J Steven Jenkins and Vance A Heron. Producing Engineering Documents Using Semantic Web Tools and DocBook. In *Proceedings of the Conference on Systems Engineering Research (CSER), March,* pages 14-16, 2007.

[19] W3C. XSL Transformations (XSLT). http://www.w3.org/TR/xslt, 1999.

[20] NXP. ADE Introduction. 2013.

[21] Helmut Kopka. LaTEX: eine Einfuehrung. Addison Wesley, 1992.

[22] Laurie Williams and Alistair Cockburn. Guest Editors' Introduction: Agile Software Development: It's about Feedback and Change. *Computer*, 36(6):39–43, 2003.

[23] Konstantin Beznosov and Philippe Kruchten. Towards Agile Security Assurance. In *Proceedings of the 2004 Workshop on New Security Paradigms*, NSPW '04, pages 47–54, New York, NY, USA, 2004. ACM.

[24] Jim Highsmith and Alistair Cockburn. Agile software development: The business of innovation. *Computer*, 34(9):120–127, 2001.

[25] W3C. XML Essentials. http://www.w3.org/standards/xml/core, 2010.

[26] W3C. XML Schema. http://www.w3.org/XML/Schema.html, 2014

[27] Microsoft. MSDN article - XML Schema Definition Tool (Xsd.exe). http://msdn.microsoft.com/en-us/library/x6c1kb0s%28v=vs.85%29.aspx, 2014.

[28] Microsoft. MSDN article - XML Schema Binding Support in the .NET Framework. http://msdn.microsoft.com/en-us/library/sh1e66zd%28v=vs.85%29.aspx, 2014.

[29] W3C. XML Schema Tutorial.http://www.w3schools.com/schema/default.asp, 2014.

[30] Heiko Koziolek. Goal, Question, Metric. In Irene Eusgeld, Felix C Freiling, and Ralf Reussner, editors, *Dependability Metrics*, volume 4909 of *Lecture Notes in Computer Science*, pages 39–42. Springer Berlin Heidelberg, 2008.

[31] Wolfgang Raschke et al. Supporting Evolving Security Models for an Agile Security Evaluation. In *Proceedings of the 1st International Workshop on Evolving Security & Requirements Engineering,* ESPRE'14, pages 31-36, 2014.