Seamless Model-based Requirements Engineering: Models, Guidelines, Tools

Sabine Maria Teufl

Technische Universität München

Institut für Informatik
der Technischen Universität München

Seamless Model-based Requirements Engineering: Models, Guidelines, Tools

Sabine Maria Teufl

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende:	Prof. Gudrun J. Klinker, Ph.D.
Prüfer der Dissertation:	
1.	Prof. Dr. Dr. h.c. Manfred Broy
2.	Prof. Dr. Barbara Paech Universität Heidelberg

Die Dissertation wurde am 23.05.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 02.09.2017 angenommen.

Abstract

Quality factors of development artifacts are characteristics that positively or negatively influence the development of a system or the system itself. Negative quality factors we call quality issues. Early detection of quality issues saves resources and leads to better products. Requirements engineering (RE) provides the means to avoid and detect quality issues in the requirements specification. In a seamless development approach, RE also aids the identification of quality issues during the processing of requirements in subsequent development artifacts. In model-based RE, models are constituents in the development process. A broad variety of model-based quality assurance (QA) techniques have been developed; each of which addresses specific aspects in the prevention and identification of quality issues in the requirements specification and in the processing of requirements. However, these techniques have been researched and evaluated in isolation; little effort has been undertaken to integrate them.

The integration of a set of model-based QA techniques into a common approach has several benefits. Combining QA techniques increases the coverage of relevant quality factors. Furthermore, the specification effort can be reduced by reusing common information between the QA techniques. The integration yields four challenges: The selection of QA techniques to address relevant quality factors; practitioners demand the specification of requirements both as unconstrained prose and using a formal representation; an integration in a seamless development approach; an adequate guidance and tool support, which is indispensable for advanced QA techniques. As both the benefits and challenges of the integration are still present when focusing on methods specific for functional requirements, this thesis narrows its scope to functional requirements.

The first contribution of this thesis is a systematic investigation of essential characteristics of a model-based RE approach to enable effective and efficient model-based QA. The development of the individual QA techniques is not part of the thesis. The second contribution is the construction of a seamless model-based RE approach called Model-based Integrated Requirements Analysis[1] (MIRA). MIRA addresses the four challenges and operationalizes the essential characteristics identified in the first contribution. MIRA combines a set of model-based specification and QA techniques and integrates them in a seamless development approach. MIRA provides guidance and tool support to address the adequacy, consistency, completeness and unambiguousness of functional requirements and the adequacy of their realization. The third contribution is the application of MIRA in case studies in the embedded software and systems domain. These case studies were conducted in collaboration with industrial partners and are based on industrial specifications. A case study of a train control system demonstrates the effective applicability of MIRA for quality assurance with concrete findings and the scalability of MIRA to the size of a system function. MIRA is designed to be extensible in order to compensate the limitation to functional requirements. A second case study in the avionics domain demonstrates this extensibility of MIRA with further types of requirements. Additional case studies indicate that MIRA is also applicable in a broader context.

[1] The term 'requirements analysis' is often used as a synonym for 'requirements engineering'.

Acknowledgements

I would like to thank all the people who supported me in many ways during the work on this thesis.

First of all, I would like to thank my supervisor, Prof. Manfred Broy, for advice and feedback on the thesis and for providing support whenever I needed it. I also would like to thank Prof. Barbara Paech for her help as second supervisor.

Furthermore, I thank all my colleagues from the fortiss research institute and the research group Software & Systems Engineering for their collaboration, interesting discussions and a pleasant working environment. In particular, I want to thank Dongyue Mou, Daniel Ratiu, Maged Khalil, Florian Hölzl, Vincent Aravantinos, Sebastian Voss, Levi Lúcio, and Bernhard Schätz from fortiss and Andreas Vogelsang, Wolfgang Böhm, Veronika Bauer, Henning Femmer, Jakob Mund, Max Junker, Jonas Eckhardt and Daniel Mendez Fernandez from the research group Software & Systems Engineering. I would like to thank all the administrators at fortiss and TUM, in particular Silke Müller, Helge Hansemann and Monika Glashauser. Special thanks to the research partner Siemens Rail and in particular to Ralf Pinger for the collaboration in our research project.

Last but not least, my friends and family who supported me in every possible way deserve a very warm thanks, especially my parents Edeltraud and Günther, my brother Toni and sister Anna, Johanna Fink and Mav.

Contents

Chapter 1

Introduction

The topic of this thesis is the development of a model-based requirements engineering approach for specifying functional requirements. The aim of the approach is to improve the quality of functional requirements and their realization in the architecture using model-based quality assurance techniques. The approach integrates existing, proven methods and models in order to address a broad range of quality issues.

1.1 Motivation

A major goal of *Requirements Engineering* (RE) is to obtain a consolidated agreement of all relevant stakeholders of a system under development on what the system should accomplish. A *requirements specification* documents the results of RE. The requirements specification is not an end in itself. The requirements specification is an input document for further development activities; the system is designed, implemented and tested. The system should conform to the requirements specification in order to obtain a system that corresponds to the stakeholders' expectations. As the requirements specification is a central document in the development process, its quality is of crucial importance.

The quality of a requirements specification can impact development activities. Through the development activities it can even impact subsequent development artifacts. A low-quality requirements specification may for example increase the development effort. An incomplete requirements specification may lead to missing functionality of the developed product. If these problems are detected late in the development process, then subsequent development artifacts may have to be revised and corrected, for example, tests and documentation. Development activities such as testing may have to be repeated. Boehm and Papaccio [BP88] claim that fixing or reworking software in late phases of the software life cycle is 50 to 200 times more expensive than in early phases. Therefore, a high quality requirements specification and a high quality transition of information contained in the requirements specification to subsequent artifacts can save resources and lead to better products.

The quality of an artifact can be captured by quality factors. A *quality factor*, or syn-

onymously called quality attribute, is a *"feature or characteristic that affects an item's quality" [ISO10]*. Quality factors with a negative impact on development activities or subsequent artifacts are called *quality issues*. *Quality Assurance* (QA) is a common means in RE to improve the quality of the system under development at the requirements level. QA can be addressed constructively and analytically. Constructive QA uses adequate means during the specification of requirements to avoid the introduction of quality issues. Analytical QA aims to detect quality issues by investigating the requirements specification and subsequent development artifacts after their creation. Analytical QA activities on the requirements specification are:

- The *analysis* aims to detect quality issues within the documented requirements that effect subsequent development activities such as inconsistencies [CA07].
- The *validation* aims to ensure that the documented requirements correspond to the actual needs of the involved stakeholders.
- The *verification* aims to ensure that a subsequent development artifact implements the documented requirements adequately.

To increase the quality of the system under development at the requirements level, the requirements specification should be designed to support these QA activities. In *model-based RE*, a broad variety of approaches has been developed to support QA. *Model-based RE approaches* provide structural rules and a vocabulary in a modeling language for the documentation of the entities, relationships, behavior, and constraints of the problem being modeled [CA07]. In the *model-based specification*, the defined structural rules and vocabulary are applied to create models. The process of creating precise models can uncover details that were missed in the initial elicitation and help to raise the level of abstraction in the description of requirements [CA07]. *Model-based QA* refers to the application of *model-based QA techniques* that make use of the structural rules and the vocabulary by applying precise rules for the interpretation of the models.

While model-based approaches promise an effective quality assurance, such an approach faces four major challenges. Model-based techniques are typically limited in their expressiveness due to a restricted syntax and semantics. A model-based RE approach needs to overcome this limitation. Quality assurance has many facets and should ensure that: quality issues in the requirements specification do not adversely affect or even impede subsequent development activities; that the documented requirements correspond to the actual needs of the involved stakeholders; and that subsequent development artifacts implement the documented requirements adequately. An operationalization of model-based RE requires guidance and tool support to apply a model-based RE approach effectively and efficiently.

The state of the art in RE is a collection of technologies like modeling languages and techniques for either requirements analysis, validation or verification. This broad variety of models and model-based methods often has been "researched and evaluated in isolation, with little knowledge of how to combine techniques effectively" [CA07].

1.2 Problem Statement

The following challenges exist for the model-based specification and QA of requirements. Isolated solutions for these challenges have been developed in research or are already applied in practice. Nonetheless, a holistic solution that addresses all of these challenges is still missing.

Challenge 1: Supporting the Textual and Formal Representation of Requirements. Many practitioners in the embedded systems domain are dissatisfied with using natural language for the specification of requirements [STP12]. One of the main reasons for the dissatisfaction is the high effort for QA. Model-based techniques, particularly formal techniques, have been proven suitable to enable an effective and efficient QA. However, practitioners "do not wish to replace natural language requirements with requirement models, but that they wish to amend them" [STP12], as natural language enables the negotiation of requirements amongst the various stakeholders [STP12]. Furthermore, formal techniques are limited in their expressiveness. RE research should provide methods enabling the joint use of natural language requirements and requirement models [STP12]. In a *heterogenous requirement specification*, requirements can be represented in different forms, for example, in natural language or formally. Where necessary, this also allows for the same requirement to be represented in more than one representation form in the requirements specification. Hence, an effective model-based RE approach for quality assurance must also recognize and support heterogeneity. Currently, few approaches provide support for the quality assurance on heterogeneous requirement specifications. These approaches provide only limited or no solutions for the remaining challenges.

Challenge 2: Covering the Analysis, Validation and Verification of Requirements. In the field of semi-formal and formal modeling languages, a broad variety of techniques and approaches has been developed for the various QA activities [Eza15]. Only few of these approaches cover all QA activities; the analysis, validation and verification of requirements. For each model-based QA technique, the requirements specification has to exhibit specific characteristics. For example, the requirements specification has to contain specific information or has to be represented using a modeling language with specific characteristics. These characteristics are quality factors with respect to quality assurance. For an effective combination of QA techniques, a systematic investigation and consolidation of these quality factors and their combined impacts on QA activities in an industrial context is required.

Challenge 3: Integrated in a Seamless Development Approach. Model-based development approaches often either do not define the artifacts that are to be created during the system development, or the artifacts are based on separate and unrelated modeling theories [BFH+10]. This impedes the transition from one artifact to another in the development process [BFH+10]; the transition is unclear and error-prone. When the transition is unclear, requirements cannot be incorporated systematically into subsequent development artifacts. This negatively impacts the verification of subsequent development artifacts against their requirements. *Seamless model-based*

development provides mechanisms for a systematic transition in the development process: a) A comprehensive system modeling theory serves as a semantic domain for the formal definition of the system under development; b) an integrated architectural model describes the detailed structure of the product (the product model) and the process to develop it; and c) a mechanism to build tools that conform to the modeling theory and allow the authoring of the product model [BFH+10].

1.2.1 Challenge 4: Providing Guidance and Tool Support for Challenges 1 - 3

Sikora et al. [STP12] showed in a study that practitioners have a strong demand for model-based approaches for requirements quality assurance. Such an approach should provide guidance about how to use models in the RE process. Participants stated that the "lack of appropriate method guidance" [STP12] is a major hindrance to the more intensive use of models in RE and, in addition, a lack of commercial tool support integrating these models in a seamless development process. Furthermore, model-based QA techniques can and often must be automated by tools to be efficiently applicable [vL09, p. 145]. A systematic investigation of the requirements from industry on such guidance and tool support is still missing.

1.2.2 Problem

> *We miss a comprehensive RE approach for the model-based specification, analysis, validation and verification of requirements for both textual and formal representations that is integrated in a seamless development approach and provides method and tool support.*

Solving this problem would help practitioners in developing high quality artifacts at early development phases. Furthermore, such a model-based RE approach would enable tool vendors to build a comprehensive tool to increase the efficiency of QA.

The main objective of this thesis is to investigate the different facets of model-based RE for quality assurance, to develop a holistic approach that incorporates these facets and to demonstrate that the resulting approach is effectively applicable in industry. As the benefits and challenges are still present when focusing on functional requirements, this thesis narrows its scope to improving the quality of *functional requirements*. The functional requirements investigated in this thesis are those requirements that define the required logical interface behavior of the system under development. Furthermore, this thesis has a scope on embedded software and systems. Summarizing the four challenges, the guiding research question for this thesis is:

> *How to specify functional requirements in a seamless model-based development approach for embedded software and systems engineering in order to enable the model-based analysis, validation and verification for both textual and formal representations?*

1.3 Solution

This thesis investigates the four challenges and constructs and evaluates a solution to these challenges in the form of a model-based RE approach that provides concrete step-by-step guidance and that is supported by a tool. The resulting approach is called *Model-based Integrated Requirements Analysis* (MIRA).

This thesis investigates and integrates three very different kinds of model that can be applied for RE. An *artifact reference structure* defines a blueprint of the contents and dependencies of the results elaborated and documented during software and systems engineering. These results are materialized in *artifacts*. A *system model* abstracts from the system under development. This thesis is based on the mathematical and logical system modeling theory FOCUS [BS01]. *Specification techniques* provide rules on a concrete syntax and semantics for the representation of parts of the requirements specification.

The MIRA approach consists of such an artifact reference structure, a guideline, and a tool.

- The *MIRA artifact reference structure* defines a set of concepts, their attributes, representation forms and dependencies to be specified during RE. Instantiating the artifact reference structure results in a requirements specification. The artifact reference structure includes the main concepts and modeling paradigms recommended by the integrated approaches.
- The *MIRA guideline* provides instructions for the requirements specification and quality assurance of functional requirements. The guideline consolidates the activities and model-based QA techniques recommended by the integrated approaches and tailors them to the MIRA artifact reference structure.
- The *MIRA tool* implements support to instantiate the artifact reference structure and to conduct the guideline. It demonstrates the feasibility to implement the MIRA approach. The tool provides automation for the model-based QA techniques.

The MIRA approach addresses the four challenges as follows:

Challenge 1 (Textual and Formal Representation). The MIRA artifact reference structure incorporates both the textual and formal representation of functional requirements. The MIRA tool provides model-based techniques for the quality assurance of both textual and formal requirements. Their application is described in the MIRA guideline.

Challenge 2 (Quality Assurance). We conducted a study to systematically investigate model-based techniques to cover the different QA activities. The implementation of the MIRA artifact reference structure as a data model in the MIRA tool provides mechanisms to check a requirements specification for syntactic aspects of completeness and consistency [MF11]. Formal specification techniques promise greater precision in the formulation of statements. Formal specification techniques additionally offer precise rules for their interpretation, thereby enabling a formal analysis. A

system model that is underlying the formal specification techniques enables a formal verification of architectural models against their functional requirements.

Challenge 3 (Seamless Development). MIRA is defined as a part of a system modeling framework that structures the artifacts developed during software and systems engineering. Based on this framework, the MIRA artifact reference structure defines how functional requirements are embedded in a seamless development approach.

Challenge 4 (Guidance and Tool Support.) In a study, we investigated the requirements for a model-based RE approach in practice. By providing a solution for these requirements, the MIRA guideline and the MIRA tool addresses the challenge for guidance and tool support.

By providing an operationalization of seamless model-based RE in a holistic approach, we aim to uncover gaps in existing approaches that address the four challenges. We also seek to identify open research questions with respect to model-based RE. This thesis discusses these gaps and, whenever possible, suggests a potential solution.

1.4 Contribution

The first part of this thesis systematically analyzes the essential characteristics that a model-based RE approach should possess in order to address the four challenges. The first study investigates model-based quality assurance. The second study investigates the requirements for a model-based RE approach in practice. The second part of this thesis introduces the MIRA approach that was developed based on the results of the studies. We used the results of the initial two studies to choose and adopt a set of established and proven RE approaches. In an explorative bottom-up method, we *integrated* the main RE concepts, modeling paradigms, model-based QA techniques and activities suggested by these approaches in the seamless model-based RE approach called MIRA. The third part of this thesis presents how the MIRA approach has been technically validated in a feasibility analysis and evaluated in a set of case studies.

1.4.1 Systematic Investigation of Model-based RE for Quality Assurance

Contribution 1: Quality Factors of a Requirements Specification that Enable Model-based Quality Assurance. The first contribution of this thesis consists of a literature study of evidence for positive impacts of quality factors on model-based quality assurance. The study aims to increase the *effectiveness* and *efficiency* of quality assurance. Effectiveness is addressed by investigating which information has to be contained in the requirements specification for the analysis, validation and verification of requirements. Efficiency is addressed by investigating constructive and automated analytical model-based QA techniques. The study focuses on qualitative

quality, as studies [MMFFE15] indicate that quantitative quality is highly dependent on project characteristics. The study result is a list of quality factors and their impact on QA activities. The model-based QA activities should support the analysis, validation and verification in order to address Challenge 2.

Contribution 2: Requirements for a Model-based RE Approach. The second contribution of this thesis is a study of the requirements for a model-based RE tool. The requirements have been elicited from scientific RE research literature and evaluated and amended in a questionnaire by RE practitioners. These requirements reinforce that a collection of RE concepts and modeling languages without a clear guidance is not sufficient for practitioners. The study confirms and details Challenges 1 to 4 by providing concrete requirements for a model-based RE approach, for example, for representation forms and QA activities.

1.4.2 Development of the Model-based RE Approach

Contribution 3: An Artifact Reference Structure for Model-based Quality Assurance. The third contribution of this thesis is the MIRA artifact reference structure. The artifact reference structure consolidates the quality factors of a requirements specification that were investigated in the first contribution. Hence, each element defined in the artifact reference structure has a clear impact on the QA activities. Due to this impact, the conformance of a requirements specification to the MIRA artifact reference structure should enable effective quality assurance (Challenge 2).

The fundamental RE concepts integrated in the MIRA approach are

- Requirement sources, the origin of all requirements
- Glossary terms that define terms used in the requirement specification
- Goals that define the intent of the systems' stakeholders
- Use cases that define user-visible system functions by describing required and undesired interactions of the system under development with its actors
- Scenarios that describe single interaction sequences
- Interface requirements that define the required and undesired behavior of the system under development on its interfaces
- Trace links that document specific dependencies within requirements and from requirements to subsequent development artifacts.

The artifact reference structure defines a textual and an adequate *formal representation* for functional requirements, i.e., for use cases, scenarios, and interface requirements, and their dependencies using the formal system modeling theory FOCUS [BS01] (Challenge 1). Due to this system modeling theory, MIRA targets embedded software and systems engineering. The artifact reference structure includes dependencies to subsequent development artifacts (Challenge 3). Thereby, the artifact reference structure covers discipline-independent requirements on the system and its subsystems as well as discipline-specific requirements on embedded software.

Contribution 4: A Guideline for the Model-based Specification and Quality Assurance of Functional Requirements. This thesis contributes the MIRA guideline that defines a coherent set of steps describing how to conduct the requirements specification and the quality assurance of functional requirements (Challenge 4). The guideline provides instructions to create a requirements specification that is conformant to the artifact reference structure. Furthermore, the guideline describes how to apply a set of model-based techniques to the requirements specification to analyze, validate and verify the functional requirements contained therein.

1.4.3 Validation and Evaluation of the Model-based RE Approach

Contribution 5: Feasibility Analysis of the MIRA Approach. The thesis contributes a feasibility analysis of the MIRA approach through the implementation in a tool. The MIRA tool implements the MIRA artifact reference structure as a data model. The data model enforces the creation of a conformant requirements specification. To automate some of the actions which are defined in the guideline, the MIRA tool offers a set of operations on the artifact reference structure. Candidates for automation in an operation are, for example, actions with a deterministic and repetitive nature. The tool automates some of the quality assurance techniques presented in the guideline. The data model is also the basis for further operations to manipulate and visualize the requirements specification. Thereby, the MIRA tool supports conducting the MIRA guideline (Challenge 4). The tool also embeds the MIRA approach in a seamless development environment (Challenge 3).

Contribution 6: Case Study on the Effective Application of the MIRA Approach. The last contribution of this thesis is a case study that evaluates the effective application of the MIRA approach in an industrial setting and demonstrates the scalability of MIRA to the size of a system function. The case study applied MIRA to the door control of an automated train control system. The case study was based on industrial specifications. The case study covers all four challenges. The case study aimed to demonstrate the effective application of MIRA. Effectiveness was measured by concrete findings in the input documents that were identified by model-based specification and QA techniques. These concrete findings confirm or extend some of the positive impacts identified in the first contribution. Another case study in the avionics domain shows that researchers can extend the MIRA artifact reference structure to other requirement types than functional requirements. The case studies and further feedback from MIRA users led to an improvement of the MIRA approach. A number of further case studies indicate that the MIRA approach is also applicable in a broader context; for software requirements, by other researchers and in a broad variety of domains.

1.5 Integrated Approaches

This section provides a brief overview of the approaches that have been integrated into MIRA. More details to the integrated approaches are provided in Chapter 2.

1.5.1 Fundamental RE Concepts and Basic Specification Techniques

Fundamental literature of often cited authors provides established practices for RE. For example, Zave and Jackson [ZJ97] provide the basics for RE in general and in particular for model-based RE. Although not very recent, the works of Zave and Jackson still remain relevant as the number of recent citations suggests. Fundamental specification and QA techniques are described in van Lamsweerde [vL09]. Pohl [Poh10] classifies and describes a set of common requirement sources, the origin of all requirements. Cockburn [Coc00] provides basics for documenting use cases and scenarios. Gotel et al. [GCHH+12b] document the fundamentals for traceability.

1.5.2 System Modeling Framework

The SPES modeling framework [PHAB12] (SPES) is a model-based software and systems engineering approach. SPES was developed in the Software Platform Embedded Systems 2020 (SPES 2020) project with 21 partners from industry and science. It defines development artifacts, their representation and their relationships for different engineering concerns and different levels of system granularity. The framework was developed to support 'seamless' engineering of embedded systems. MIRA is based on the concepts, relationships, engineering concerns and system level granularities defined in SPES.

1.5.3 Formal System Modeling Theory

FOCUS [BS01, Bro10b, Bro13a] is a formal system modeling theory especially suitable for functional requirements of discrete, multifunctional systems. FOCUS provides a theory and formal specification techniques to describe a system under development in terms of its logical *input/output behavior*. The system behavior is described in terms of histories of messages that are exchanged at the interfaces of the system. Both the interaction of the system with its environment and internal interactions of system components can be described. Furthermore, FOCUS provides the mathematical and logical background for the refinement and verification of requirements. FOCUS is the basis for the formal representation of requirements in MIRA.

1.5.4 Guidance

MIRA builds on AutoRAID [SFGP05], [Sch09, p. 21ff]. AutoRAID provides an artifact reference structure for requirements that integrates requirements tightly with model elements of subsequent development artifacts. A guideline supports the requirements engineer in the textual specification of requirements and their integration. AutoRAID does not provide a formal representation for requirements that is independent of subsequent design models. MIRA adopts the guideline provided by AutoRAID.

Cimatti [CRST09, CRST13] provides an RE modeling theory for hybrid systems. It includes a modeling language, a set of analysis methods, an artifact reference structure

for the requirements specification and a guideline for the formalization and analysis of requirements. Whilst the modeling language, the analysis techniques and the artifact reference structure differ to some extent to those integrated in MIRA, the guideline is incorporated.

Broy [Bro10b] proposes an approach to structure and formalize functional requirements for multifunctional systems based on the formal modeling theory FOCUS. MIRA adopts this guideline for the formal specification of functional requirements.

1.5.5 Tool Support

AutoFOCUS3 [HF10, AVT+15] is a scientific open source tool for model-based development. It instantiates a subset of the FOCUS modeling theory and thereby provides a set of graphical specification techniques. Furthermore, AutoFOCUS3 provides a wide range of formal QA techniques. AutoFOCUS3 is conformant to the SPES modeling framework. MIRA is implemented as a plug-in to AutoFOCUS3. MIRA facilitates the application of the specification and QA techniques provided by AutoFOCUS3 to RE.

1.6 Running Example

Whenever necessary, examples illustrate models and model-based techniques. The running example is a traffic light controller that is used to control all lights of a simple pedestrian crossing with a pedestrian light and a traffic light, as illustrated in Figure 1.1. A model of the example is included in the release 2.7 of AutoFOCUS3 with the name 'Simple Traffic Light AF3 Tour Example'.

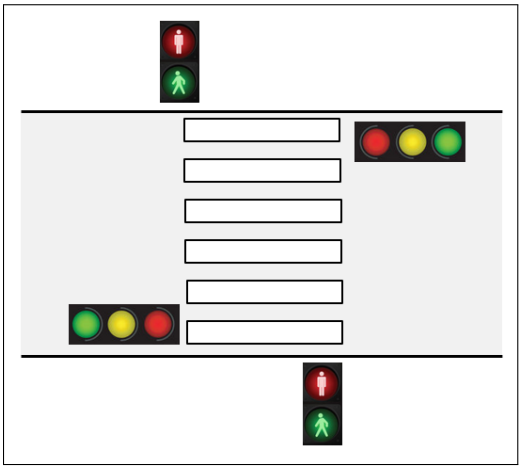

Figure 1.1: Crossing with pedestrian lights and traffic lights

1.7 Outline

Figure 1.2 illustrates the outline of the main body of this thesis. Chapter 2 presents the relevant background knowledge. Chapter 3 provides an overview of model-based RE approaches that are related to the approach developed in this thesis. Chapter 4 investigates model-based quality assurance in requirements engineering. In Chapter 5, the requirements for a model-based RE tool are investigated. In Chapter 6, the MIRA artifact reference structure is presented that results from the investigations. Chapter 7 constructs the MIRA guideline based on the integrated approaches. The MIRA tool is presented in Chapter 8. In Chapter 9, the case studies conducted with MIRA are presented. Finally, Chapter 10 concludes the thesis by summarizing the contributions of this thesis and the open research questions.

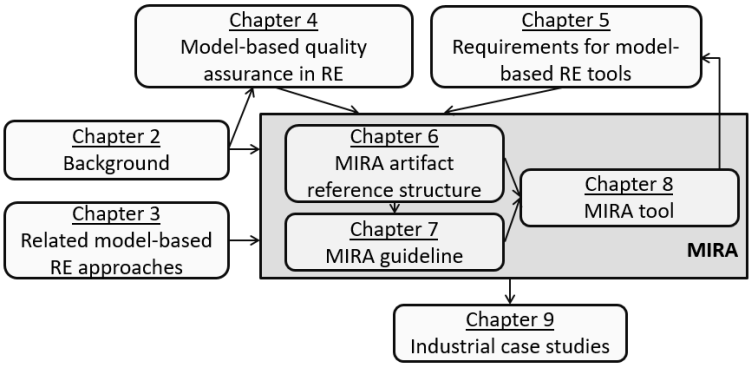

Figure 1.2: Outline of the main body of this thesis; the arrows indicate how the chapters relate to each other

Previously Published Material. Parts of the contributions presented in this thesis have been published in [TMR13], [TKM13], [TBP14], [BJV+14], [VEH+14], [TH15], [RTVH15], and [AVT+15].

Chapter 2

Background

This chapter introduces to the fundamental concepts and terms used in this thesis. Relevant background knowledge on RE is provided in Section 2.1. Section 2.2 introduces the background knowledge on quality and quality assurance. Section 2.3 introduces the terms and modeling paradigms for model-based RE used in this thesis.

Contents

2.1 Requirements Engineering

Requirements engineering (RE) is a development phase in the lifecycle of a system. RE is concerned with understanding and defining the problem that needs to be solved by the system under development, more precisely, "*what* problem should be solved, *why* such a problem needs to be solved, and *who* should be involved in the responsibility of solving that problem" [vL09].

Jackson [Jac95] introduced the concepts of world and machine to explore the role of RE. The problem to be solved pertains to an organizational, technical or physical *world*. The aim of the developed solution, called *machine* by Jackson [Jac95], is to solve a problem of the world. This work focuses on the development of embedded systems. SPES [BDH+12] defines embedded systems as "microcontrollers, which are connected to complete systems via sensors, actors, operator controls, and communication devices, while interworking diversely with their environment, offering a variety of functions by comprehensive software." RE for embedded systems spans over different levels of granularity, from detailed hardware and software requirements for the embedded system to (hardware- and software-independent) requirements for the system that is controlled by the embedded system. Systems controlled by embedded systems may range from simple systems such as a coffee machine to complex systems like a train. The machine is the system under development at a particular level

of granularity. The world is the system context according to the level of granularity. For example, if the machine is embedded software, the world includes the execution platform of the software. The world of an embedded system includes the controlled system.

The world and the machine interact with each other [Jac95] by sharing 'phenomena'. These shared phenomena define the interface through which the machine interacts with the world. The machine monitors some of the phenomena of the world while controlling others in order to solve the problem. For example, the passenger exchange of a train in a train station are phenomena in the world. The train speed, its exact stopping point in the station and the train door status can be measured and shared with the machine, a control software of the train. The door commands for the doors can then be calculated in the control software and shared with the world to control the train doors. The example is visualized in Figure 2.1. RE is concerned with the machine's effect on the world and the assumptions made about that world [vL09, p. 4], especially with the shared phenomena of world and machine.

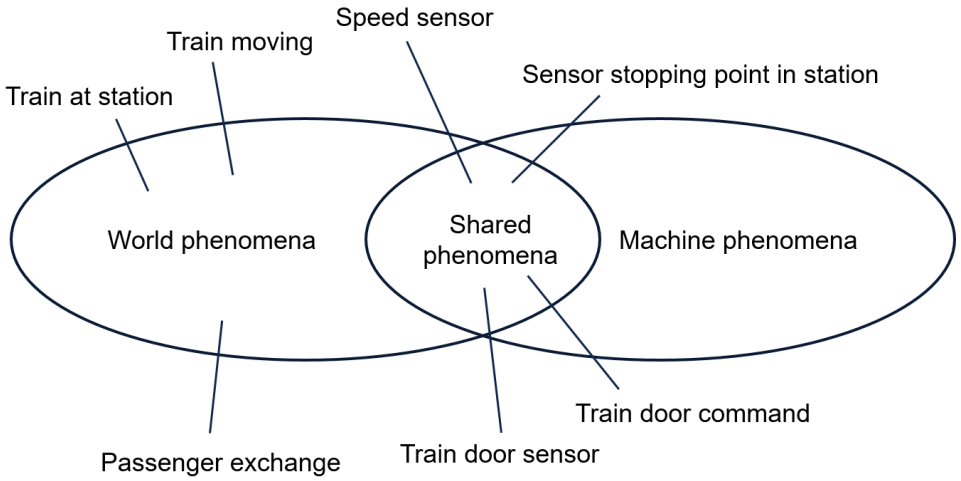

Figure 2.1: The world and the machine (following Jackson [Jac95]) on an example of a train control software

Requirements describe the problem to be solved *solution-neutral* in terms of properties of the world, or *solution-oriented* in terms of shared properties of the world and the machine. Conventionally, solution-oriented requirements are distinguished into *functional requirements* that define what the system should do and *extra-functional requirements* (also called non-functional requirements) that define how the system should do something [RR06, KS98]. Functional requirements "are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations" [Som11]. Functional requirements express that "a system offers a particular functional feature such that it can be used for a certain purpose (use case), or a system's function (a specific instance of a functional feature) has a property formulated in terms of the system's interface behavior" [Bro15a]. Functional requirements include required and undesired interface behavior of the system under development as well as wanted and unwanted interaction of the system with its environment.

This thesis applies the FOCUS system modeling theory as presented in Section 2.3.5 to rigorously define and express functional requirements. FOCUS has the means to model required histories of interactions between a system and its environment. FOCUS facilitates the expression of requirements on the syntactic system interface (the system inputs and outputs) and requirements on the system interface behavior (which history of system inputs leads to which histories of system responses). The correct system response may even be nondeterministic, if the set of histories of system responses has more than one element. Extensions of the modeling theory [Neu12] facilitate the expression of quantitative interface behavior by assigning probabilities to each possible system response. These extensions can be used to express requirements on the quality of the system under development, such as availability [JN12]. These extensions are outside the scope of this work.

The main artifact developed in RE is the *requirements specification*. The requirements specification comprises requirements and additional information about the problem to be solved. For example, some standard or some activity that is executed on the specification may require to document further information in addition to the requirements in the requirements specification. According to van Lamsweerde [vL09, p. 17], a requirements specification can contain descriptive and prescriptive statements. *Descriptive* statements state properties and describe assumptions about the world. These statements are indicative, so they hold regardless of the system behavior, for example, physical constraints. *Prescriptive* statements state desirable properties of the system under development. They are optative, so they may hold or not, depending on the system behavior. Such statements need to be enforced by components of the system. Prescriptive statements may need to be negotiated, weakened and changed or alternatives may be found. The prescriptive statements constitute the actual requirements. By contrast, descriptive statements cannot be negotiated, weakened, changed, alternatives cannot be found.

Different *RE phases* can be distinguished according to distinct activities to be conducted, see also Cheng and Atlee [CA07], Kotonya and Sommerville [KS98] and van Lamsweerde [vL09]:

Domain understanding and elicitation. The system-as-is is studied within its social and technical context. The stakeholders of the system are identified and the requirements imposed by the stakeholders are elicited. RE literature [KS98, vL09] proposes a variety of techniques such as interviews, questionnaires, or workshops for eliciting requirements from persons. Requirements stemming from relevant laws, certifications and standards have to be identified and existing documentation on the system-as-is studied. The elicitation may lead to alternative or even conflicting requirements.

Evaluation and negotiation. Requirements are elicited from different sources and viewpoints. These requirements reflect the needs of different stakeholders that might be conflicting. This set of requirements has to be consolidated and agreements have to be achieved. The requirements *analysis* consolidates the different views on the system under development and thereby identifies quality issues in the requirements specification resulting from these different views, for example, *inconsistent* requirements.

Specification. The specification phase is concerned with creating the requirements specification. The information to be contained in a requirements specification is *documented*. The information to be contained in the requirements specification can be

documented informally as prose or *formalized* by applying a semi-formal or formal modeling language. Documenting associations between development artifacts, for example, between requirements, is called *tracing*.

Validation and Verification. The requirements specification has to be *validated* in order to ensure that the documented requirements correspond to the actual stakeholder needs. The ISO 29148 [ISO11b] defines validation as "confirmation by examination that requirements (individually and as a set) define the right system as intended by the stakeholders". The aim of *verification* is to ensure that subsequent development artifacts satisfy the requirements. The IEEE 12207 [IEE08] defines verification as "confirmation, through the provision of objective evidence, that specified requirements have been fulfilled".

As Figure 2.2 depicts, these phases are often conducted iteratively.

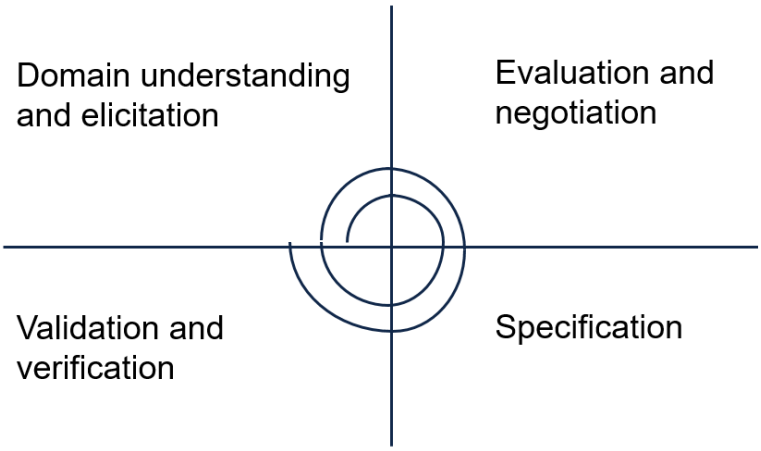

Figure 2.2: Iterations through the RE phases, see also van Lamsweerde [vL09]

Over all these phases requirements have to be managed. *Requirements management* is the task of managing requirements, especially changes to the requirements [KS98], [vL09]. Requirements management should keep track of these changes. Furthermore, requirements management should ensure that changes are made to the requirements document in a controlled way by change control and change impact assessment [KS98, p. 11].

This work focuses on activities of the late phases of RE, the specification, analysis, validation and verification of functional requirements. It investigates the information to be contained in the requirements specification that is necessary for conducting these activities.

2.2 Quality Assurance

Quality assurance is defined as *"1. a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements 2. a set of activities designed to evaluate the process by which products are developed or manufactured. 3. the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity*

will fulfill requirements for quality. 4. part of quality management focused on providing confidence that quality requirements will be fulfilled" [ISO10]. Hence, quality assurance can be product-centered, process-centered or focus on quality requirements. This thesis investigates product-centered quality assurance. The products in scope of this thesis are the requirements specification and subsequent development artifacts. In this thesis, RE quality assurance seeks to improve the quality of the system under development by improving the quality of the requirements specification and by assuring a correct processing of the requirements.

Analytical Quality Assurance aims to detect quality issues by investigating existing artifacts. It comprises activities performed only on the requirements specification and activities that are performed on the requirements specification and other development artifacts. The activities concerned with quality assurance are analysis, validation and verification. Requirements management provides the means to change the requirements specification in a controlled way to resolve quality issues. Figure 2.3 provides an overview of the analytical quality assurance activities in the scope of this work and the RE phases to which they belong. Analytical techniques differ in their degrees of automation. A technique is *manual* when conducted by a human; it is *semi-automated* when it is conducted by a human and a machine; it is *automated* when it is conducted by a machine.

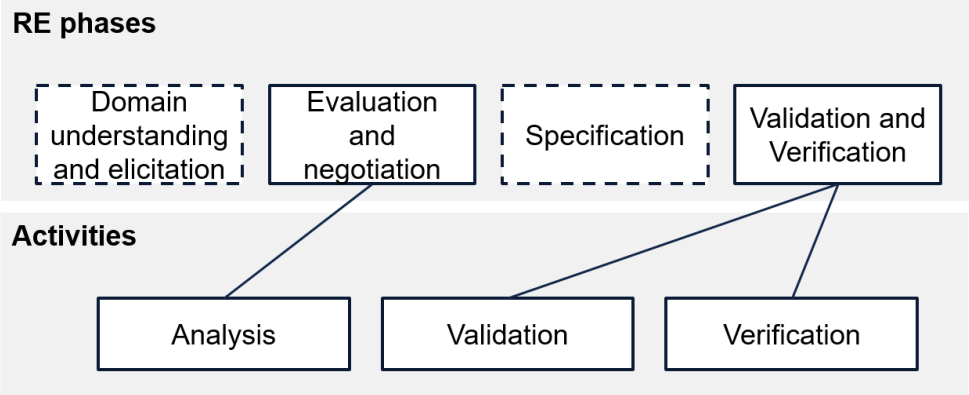

Figure 2.3: Activities in the RE phases in scope of this thesis for analytical quality assurance

Constructive quality assurance aims to achieve desired characteristics of an artifact and to prevent the introduction of quality issues during the development of an artifact. Constructive quality assurance can be applied during all phases of the development of the requirements specification. This work specifically focuses on rules and guidelines to improve the quality of the requirements specification constructively during the specification phase. Figure 2.4 summarizes the activities of the specification phase in the scope of this work for constructive quality assurance activities.

In an activity-based view, the quality of an artifact is not an end in itself. The quality of an artifact can be investigated from the activities performed on it. A quality factor of a requirements specification "impacts the stakeholder's ability to perform his specific activity efficiently and effectively" [FMF15]. The *impact* of a quality factor of the requirements specification on these activities can be positive or negative. Hence, quality factors can be divided into *quality characteristics* with a positive impact and its opposite, *quality issues* with a negative impact. Through the impacts on development

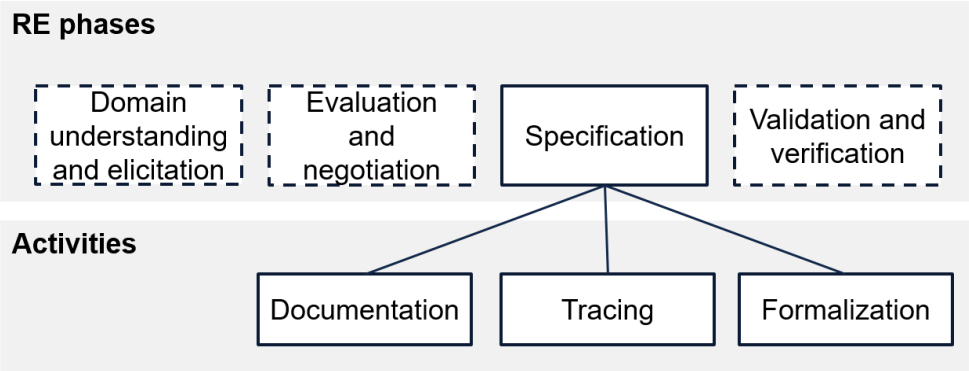

Figure 2.4: Activities in the RE phases in scope of this thesis for constructive quality assurance

activities, the quality factors can have an effect on the system under development.

This thesis investigates the constructive and analytical quality assurance of functional requirements.

2.3 Model-based Requirements Engineering

Following the main characteristics of a *model* defined by Stachowiak [Sta73], models are developed for pragmatic reasons and are a reduction of the original that they represent. Models should be developed for a particular purpose, even if they are sometimes only valid within a particular time interval or restricted to particular mental or actual operations. A model is an abstraction of the original that eliminates unnecessary details and concentrates on the core problem [Bro15b].

A *modeling language* defines the elements and their relations captured in a model by a set of concepts and their dependencies. A modeling language defines the syntax and semantics of the actual model. The model results from the instantiation of a modeling language. A modeling language can comprise different *specification techniques* to define the syntax of particular parts of the model. The model or parts of it can be represented *graphically* in diagrams, in *tabular* form, or *textually*. Modeling languages can be distinguished by their *degree of formality* [BS01, p. 9]. *Informal* techniques document requirements as prose in unrestricted natural language. *Semi-formal* techniques are based on modeling languages that "seemingly are formal, but lack a precisely defined syntax or contain constructs with an unclear semantics" [BS01, p. 9]. *Formal* techniques require modeling languages with formal syntax and formal semantics, based on a rigorous mathematical theory. To support the refinement over different levels of abstraction, a formal modeling language may offer support for logical deduction of implementations from specifications and, in addition, a mathematical notion of refinement and a calculus for the verification of refinement steps.

Model-based engineering is an approach to the development of software and systems where models are seen as constituents of the development process. In model-based engineering, models are used to support development activities. These activities may, but not necessary have to include the transformation of a model to software

code that is centric to model-driven development [FR07]. Model-based engineering can also be used to gain a better understanding of the system to be developed, to successively increase the understanding of the system under development. Model-based engineering promises to achieve significant improvements in both productivity and quality. It can support the engineer in managing or even reducing system complexity by removing unnecessary details. Finally, model-based engineering can provide models that enable advanced model-based techniques for quality assurance.

Model-based engineering is most often associated with the design or implementation phase of the systems and software development. However, greater rigor can also be introduced into RE by applying model-based engineering principles leading to an increase of structure and precision in the requirements specification [Sch09, p. 21]. In turn, structure and precision facilitate sophisticated analysis techniques earlier in the development process, increasing the quality and efficiency of the development process.

In *model-based requirements engineering*, models can support RE activities and those activities that use the models developed during RE as an input. Models can support all RE phases. For example, domain modeling [Bro13b] and goal modeling [vL01] provide support for domain understanding and elicitation. As soon as the first draft of the requirements specification is documented, models may help to analyze the specification, for instance, in order to detect inconsistencies between stakeholder intentions. Another potential use of models is to ease the understanding and communication of requirements. As detailed in Section 2.1, this thesis investigates the late RE phases and models to support these phases. This thesis uses model-based requirements engineering in order to improve the effectiveness and efficiency of quality assurance.

We investigate three complementary forms of models, an artifact reference structure for the requirements specification, specification techniques and a system model for seamless model-based development. The *artifact reference structure* defines a blueprint of the requirements specification. *Specification techniques* provide the rules for representing the elements defined in the artifact reference structure. *Seamless model-based development* provides abstractions and views of a system under development together with a system modeling theory that facilitates the definition and analysis of the RE artifact reference structure with respect to the system under development.

As many current approaches either focus on artifact-orientation, specification techniques or a system modeling theory, the modeling paradigms are introduced separately. The model-based RE approach presented in this thesis integrates these modeling paradigms into a holistic approach.

2.3.1 Artifact Reference Structures in Requirements Engineering

Artifact-orientation establishes a blueprint of the created RE results, their contents, and their dependencies. Artifact-orientation abstracts from the activities and processes that dictate how to undertake RE in order to create results [MFPKB10]. Artifact-orientation concentrates on the method outcomes in order to specify what has to be achieved [MFP14].

An *artifact* is an input or output of a development phase that has value to someone

involved in that phase [MF11]. An artifact can be described by the following characteristics [MFBT]:

The *physical representation* is determined by the data carrier. The information stored in an artifact can be for example stored on paper or in electronic form (in files or databases); the same information can have different physical representations, for example a file and its printout.

The *syntactic structure* can be distinguished in several formats such as natural language, formal language, images, tables and diagrams, or a mixture of these. "An artefact can be a composition of a number of sub-artefacts, being artefacts themselves. This composite structure can be described by a grammar or a meta model. As an example, consider the structure of a specification document by chapters and sub-chapters, which may be represented by a table of content. For the composition of artefacts, we need to consider (1) that there exists a least granularity of (sub-) artefacts, and (2) that sub-artefacts have relations and dependencies" [MFBT].

"The *semantic content* of an artefact represents its meaning. We distinguish two ways of capturing and interpreting the content; a pragmatic way where we provide a real-world reference and understand the content as statements on the real world, and a way where we interpret the content as a reference to the semantic theory via ontologies, mathematical system models, and so on. That is, if a semantic universe exists onto which the content can be mapped, then the relationships between the syntactic and the semantic content of artefacts and their sub-artefacts can be formalized including consistency, redundancy, and dependency. Finally, the semantic content of the artefacts might not be understood without additional context information" [MFBT].

This thesis investigates the requirements specification as the main artifact, its syntactic structure and semantic content. The physical representation is not relevant for this thesis. An *artifact reference structure* provides the means to define the syntactic structure and the semantic content of an artifact from an RE perspective grounded in the concepts and processes of the RE domain. The following elements define an artifact reference structure as described in [MF11, MFP14]:

A set of *content items* organizes the artifact. Content items categorize the contents of an artifact by responsibilities and tasks; a content item should be the outcome of one task and have a single responsibility. For example, content items can be used to divide a requirements specification into chapters. Hence, content items map the artifact to the RE perspective as they are defined according to the RE processes.

For each content item, *RE concepts* are defined that represent the concern of this content item. Concepts have a specific *type*. Concepts can be hierarchically decomposed to concept items. A differentiation of concept items is made if different items of a concept can be described with different techniques. For example, the concept 'use case' has concept items 'scenario' and 'scenario steps'. Concepts can be detailed by attributes.

The *syntax* of each concept can be defined by various specification techniques with different degrees of formality.

Trace links denote *dependencies* between artifacts, content items and concepts. Trace rules can pre-define trace links, their semantics and representation.

A user that instantiates the artifact reference structure developed in the course of this thesis produces a graph of the requirements specification. The nodes of this graph are the objects that instantiate the RE concepts. The edges of the graph are the references to other objects documented in the trace links. Whenever this thesis uses the term *requirements specification*, it refers to this graph. In order to obtain a *requirements specification document*, a user has to transform the graph or a subset of it into a document, adding an outline with chapters and sub-chapters.

2.3.2 Specification Techniques in Requirements Engineering

In RE, a variety of techniques have been proposed to represent the information to be documented for the various RE concepts. A structured description constrains and guides the documentation, thereby providing greater rigor compared to an unconstrained documentation. In the following, specification techniques are presented that are used in this work.

Semi-formal Specification. *Templates* provide named fields for the attributes of an RE concept. Templates define unrestricted fields to fill with natural text or provide more guidance through controlling the content by a selection from a predefined list.

Controlled natural language (CNL) is a way to bridge the gap between a natural language and a formal language and can mediate between these languages [Sch10]. Requirements that are represented using CNL are *semi-formal requirements*. CNLs restrict the syntax and/or the semantics of a sentence. A CNL that restricts both syntax and semantics may even define a formal specification language (as described in more detail below). CNLs can be transformed directly to templates, as an example shows: The template in Figure 2.5 corresponds to the following CNL pattern [MW10]
<preconditions> <optional trigger> the <system name> shall <system response>

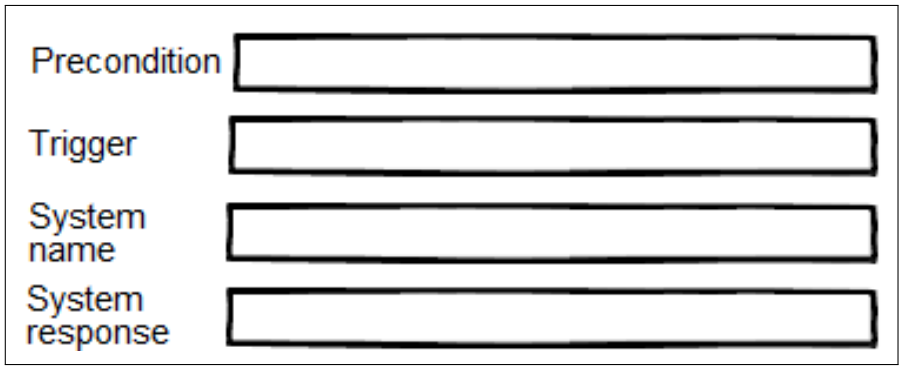

Figure 2.5: Template based on a pattern [MW10] for CNL

Formal Specification. Formal specification techniques can be applied to RE concepts and their dependencies. For instance, a *formal requirement* is a requirement that is represented using a formal specification technique. Formal specification techniques can be distinguished into two categories: Formal description techniques (FT) have "a formally defined syntax and a semantics expressed in well-understood mathematical notation" [BS01, p. 9]. Examples for these techniques are I/O assertions

[BS01], state automata [Bro10b], or message sequence charts (MSC) [ITU11a]. Formal description and development techniques (FDDT) "differ from the FDTs in their support for logical deduction of implementations from specifications. FDDTs contain one or several FDTs for specification purposes: for example, one FDT for requirements capture and another FDT for design. FDDTs offer, in addition, a mathematical notion of refinement and a calculus for the verification of refinement steps" [BS01, p. 9].

2.3.3 Seamless Model-based Development

Seamless model-based development of embedded software-intensive systems enforces a deep, coherent and comprehensive integration of models and tools in order to decrease redundancy, inconsistency and increase automation [BFH+10]. According to Broy et al. [BFH+10], this can be achieved by 1) integrating a comprehensive system modeling theory that serves as a semantic domain for the models, 2) an integrated architectural model that describes the detailed structure of the product (the product model) as well as the process to develop it, and 3) a mechanism to build tools that conform to the modeling theory and allow the authoring of the product model according to the defined process. Benefits results from their use through the entire product development in a seamless way. "For instance, requirements are the inputs for an initial system design and for test case generation. This workflow requires a deep integration of the requirements, the system design, and the tests" [BFH+10]. "Instead of working with isolated models, engineers access via dedicated views a common model repository that explicitly stores the overall product model. All required views are formally defined and based on one comprehensive modeling theory, which enables the construction and unambiguous semantic interpretation of the product architecture" [BFH+10].

An architectural model provides viewpoints on and abstraction layers of the system. *Viewpoints* [BDH+12] separate the different concerns of stakeholders during the engineering process. A viewpoint serves as a construct for managing the artifacts related to the different stakeholders of the engineering process. *Abstraction layers* [BDH+12] facilitate the definition of a system at different levels of granularity.

A *system modeling theory* provides an abstraction of the system under development with clearly defined syntax and semantics. The system modeling theory provides the theoretical basis for the formalization of all artifacts produced during the development of a system [BFH+10]. A system modeling theory suitable for seamless model-based development should provide concepts to model the different viewpoints of a system under development at different levels of abstraction. Furthermore, it should be able to describe the dependencies between different viewpoints and levels of abstraction in order to support a systematic development and verification/test of the system under development. To reduce the complexity of the system development, a suitable system modeling theory should incorporate mechanisms to capture a decomposition of the system into subsystems and a composition of subsystems to a system. Formalizing requirements determines its level of abstraction and eliminates the ambiguity of prose. A formal modeling language enables a precise specification of requirements [Bro13a].

2.3.4 SPES System Modeling Framework

The SPES 2020 project [PHAB12] developed the *SPES modeling framework* for software and systems engineering that integrates and consolidates existing model-based approaches for embedded systems.

Two prominent software engineering principles incorporated in this framework are *viewpoints* and *abstraction layers* for software and systems engineering. The *SPES matrix* visualizes the relation of viewpoints and abstraction layers as shown in Figure 2.6.

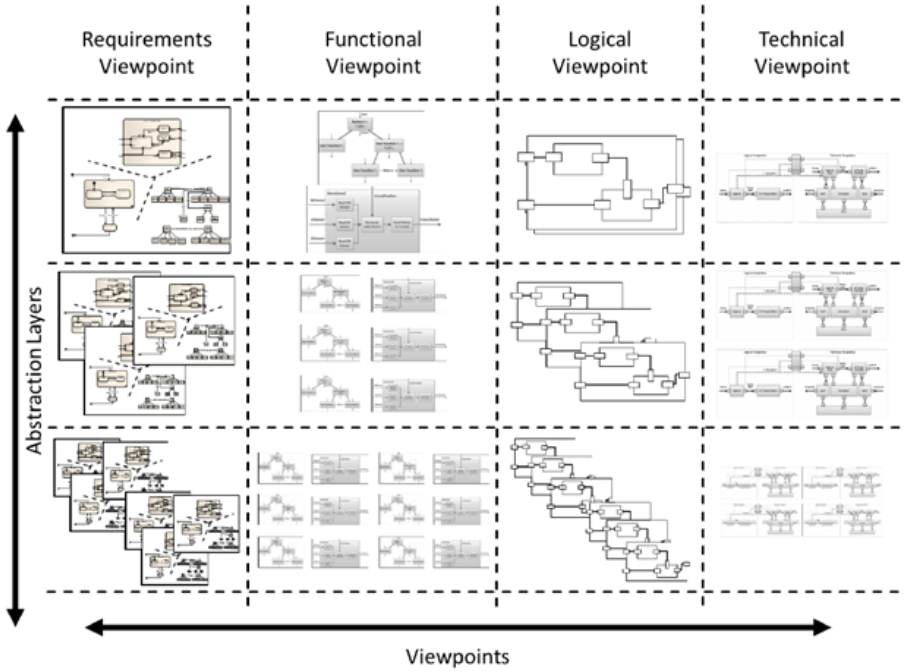

Figure 2.6: The SPES matrix (taken from [BDH+12])

Viewpoints separate the different concerns of stakeholders during the engineering process. A viewpoint serves as a construct for managing the artifacts related to the different stakeholders of the engineering process. The SPES 2020 modeling framework recommends distinguishing four viewpoints in the development process [BDH+12, DTW12]:

Requirements viewpoint defines the concepts and techniques for systematic RE.

Functional viewpoint defines the concepts and techniques to specify and model the system functions and their relationships based on the functional requirements in a *functional architecture*.

Logical viewpoint defines the concepts and techniques required to decompose the system functions into a system architecture of logical components. The resulting *logical architecture* is based on the functional viewpoint and incorporates the quality requirements.

Technical viewpoint defines the concepts and techniques required to detail the logical architecture into a *physical architecture* that, amongst other things, specifies

the hardware components of the system and the deployment of the software on those components. The technical viewpoint implements the requirements on the physical architecture.

This list of viewpoints is not exhaustive; some application scenarios require additional viewpoints.

Abstraction layers [BDH+12] facilitate the definition of an embedded system at different levels of granularity. In the development of complex systems, it is common practice to decompose a system into parts in order to tackle system complexity. The different layers of the decomposition are called abstraction layers. A *system* is decomposed into *subsystems*, the lowest level is called *implementation* or *construction*. The concrete abstraction layers chosen for a particular system vary over the diverse application domains. Whenever the scope of the system development changes to a lower level of decomposition, each of the subsystems becomes a system under development, that either itself has a decomposition into subsystems or is a software or hardware component.

For the requirements viewpoint, SPES recommends a set of RE concepts and specification techniques. A strict separation between the *solution-neutral* intentions of the stakeholders and *solution-oriented* requirements is made. The RE concepts suggested by SPES are:

Context A context model describes the part of the environment that influences or is influenced by the system under development.

Goal A goal model documents the intentions of the system stakeholders.

Scenario Examples of interactions of the system with its environment can be specified as scenarios.

Solution-oriented requirement is a solution-specific description of behavior, operations, and the information structure of the developed solution concept.

Following the recommendations of SPES [DTW12], at each abstraction layer the same RE concepts are developed in the requirements viewpoint. The RE concepts provided by the SPES modeling framework are domain-independent, but specific for embedded systems. The SPES modeling framework gives suggestions to represent RE concepts and their dependencies as diagrams, but does not provide a concise definition which specification technique to use for which RE concept.

This work investigates the RE concepts suggested by SPES for their impact on quality assurance and proposes precise specification techniques for their representation.

2.3.5 System Modeling Theory FOCUS

FOCUS provides a system model for discrete systems. A *discrete system* is defined as a system with clear cut boundaries that "interacts with its environment over this boundary by exchanging messages representing discrete events" [Bro13a]. FOCUS provides a formal technique to represent *functional requirements*, techniques to model the interfaces and interface behavior of a system in a *system design* and furthermore a technique to establish a *formal refinement specification*, both between requirements, and between requirements and system design. Refinement here means to add in-

formation to an artifact, for example, in order to derive the design of a system from its requirements. Furthermore, FOCUS provides means for the *formal verification* of requirements based on the formal refinement specification. The mathematical foundations provided by FOCUS are introduced in Broy and Stølen [BS01].

FOCUS comprises logical (as in set theory) and probabilistic behavior models to describe interface behavior [Bro15a]. This thesis uses the logical behavior models as the formal foundation for the model-based RE approach developed in this work. Probabilistic extensions of the FOCUS modeling theory facilitate to capture and express specific kinds of system quality, for example, performance and reliability, and corresponding quality requirements in terms of system interface behavior [Bro15a]. Probabilistic extensions are not in scope of this work.

The system modeling theory FOCUS was chosen for two reasons. Firstly, the modeling theory has many relevant characteristics that are necessary in this work; it provides a theory for the formal representation of functional requirements in embedded systems based on a formal system model. Additionally, it has been proven especially suitable to model embedded systems from different views and at different levels of abstraction; it provides the means for a logical description of distributed interactive systems. The second reason to select FOCUS is more pragmatic. The theory is widely used within the author's research group. FOCUS is the foundation for the research tool AutoFOCUS3 (introduced in Section 2.3.6) developed within that group. AutoFOCUS3 instantiates the FOCUS system modeling theory and provides a set of formal quality assurance techniques. Tool support for the model-based RE approach developed in this thesis has been implemented as a plug-in for AutoFOCUS3. This enables the use of these quality assurance techniques for RE. Other system modeling theories with similar characteristics may be equally suitable for the model-based RE approach. Investigating alternative theories is outside the scope of this work.

In the following, the modeling theory FOCUS is introduced based on the publications [BS01, Bro10b, Bro13a]. All definitions and formulas are literal quotes and not labeled as such to aid readability.

2.3.5.1 FOCUS Basics

In FOCUS, systems are defined by their interfaces. The syntactic interface is defined by its *input ports* and *output ports*. *Channels* connect interfaces via the ports and exchange streams of *messages*. Each instance of sending or receiving a message is a discrete event. *Data types* are assigned to each channel including the input and output ports to restrict the messages to be transmitted via the channels. The semantic interface of a system represents its black box behavior, also called *interface behavior*. The behavior is modeled as a function. This function maps streams of messages given on the input ports to streams of messages given on the output ports of the system.

A *data type* is defined by a set of values. *Elementary types* are for example the Boolean data type $\mathbb{B}$ and the set of natural numbers $\mathbb{N}$. Furthermore, *enumeration types* T with values $e_1, \ldots, e_n$ may be introduced through type declaration of the form

$$type\ T = e_1|...|e_n$$

A *syntactic interface* is a set of typed *input ports* I and typed *output ports* O. The syn-

tactic interface is denoted by

$$(I \blacktriangleright O)$$

Given a message set M of data elements of type T. The set of finite sequences of messages over M is denoted by $M^\star$. By $(M^\star)^\infty$ we denote the set of timed streams.

For a *timed stream* s in each time interval $t \in \mathbb{N} \setminus \{0\}$ a sequence $s(t)$ of messages is given. A timed stream s of type T is represented by a function

$$s : \mathbb{N} \setminus \{0\} \to M^\star$$

In each time interval t an arbitrary, but finite number of messages may be communicated.

Let C be a set of typed channels. A *total channel history* is a mapping

$$x : C \to (M^\star)^\infty$$

For each channel $c \in C$ a stream $x(c)$ is of type $T(c)$. We denote the set of all channel histories for the channel set C by $\overrightarrow{C}$. $\wp(\overrightarrow{C})$ denotes the power set of $\overrightarrow{C}$. The function F represents an *I/O-behavior* of a system

$$F : \overrightarrow{I} \to \wp(\overrightarrow{O})$$

By $IF[I \blacktriangleright O]$ we denote the set of all (total and partial) I/O behaviors with syntactic interface $(I \blacktriangleright O)$ and by IF the set of all I/O behaviors. The black box behavior, also called *interface behavior*, is given by an I/O-behavior.

An I/O-behavior $F \in IF[I \blacktriangleright O]$ can be specified by a formula in predicate logic, called interface assertion, with the channels as logical identifiers for streams. It formalizes the interface behavior of a system in a descriptive logical style.

Definition. Given a syntactic interface $(I \blacktriangleright O)$ with a set I of typed input channels and a set O of typed output channels, an *interface assertion* is a logical formula with the channel identifiers in I and O as free logical variables denoting streams of the respective types.

2.3.5.2 Formal System Specification by Interface Assertions

The formal system specification of a system consists of distinct views on the system, following the viewpoints defined in the SPES matrix (see Section 2.3.4).

Requirements Viewpoint. FOCUS understands functional requirements at the system level as a set

$$A = \{A_i : 1 \leq i \leq m\}$$

of logical assertions A_i about the interface behavior of the system. Each assertion A_i describes a property that the system must fulfill. With respect to the system modeling theory, an assertion defines required observations of behavior between a system and its environment. The observations are defined over the channel histories of a system:

$$A_i : \overrightarrow{I} \times \overrightarrow{O} \to \mathbb{B}$$

The system interface behavior F is then specified by the functional requirements. The requirement specification of a system S with the expected functionality $F \in IF[I \blacktriangleright O]$ consists of the description of its syntactic interface $(I \blacktriangleright O)$ and of an interface assertion A. The requirements can be composed into one large assertion:

$$\wedge\{A_i : 1 \leq i \leq m\}$$

Functional Viewpoint. The functional viewpoint provides a consolidated view on the functional requirements of a system. In a multifunctional system, the system can be decomposed according to its functionality into hierarchical sub-*functions*, also called services. The *function hierarchy* structures the functional requirements for that multifunctional system in a structured specification.

A system S with syntactic interface $(I \blacktriangleright O)$ and interface assertions $\wedge A$ can be decomposed a set W of sub-systems F_w with $w \in W$ with syntactic interface $(I_w \blacktriangleright O_w)$ and specifications by interface assertions B_w such that

$$\wedge\{B_w : w \in W\} \Rightarrow \wedge A$$

Each sub-system assertion B_w specifies a specific sub-function of the system.

The functional specification of the system S in terms of its subfunctions is captured by a set of assertions

$$B = \{B_w : w \in W\}$$

The assertion $\wedge B$ is the *functional specification* of the behavior of the sub-functions of the system.

Logical Viewpoint. A system S may be decomposed into a *logical* sub-system architecture. This decomposition yields a set K of subsystems S_k with syntactic interface $(I_k \blacktriangleright O_k)$. The architecture is specified by interface assertions C_k for each subsystem $k \in K$. The decomposition is governed by the principle to decompose a system into an architecture consisting of a number of sub-systems called *components*, which can be implemented and deployed independently.

Then by

$$C = \{C_k : k \in K\}$$

the set of specifications of the subsystems is denoted. The assertion $\wedge C$ is the specification of the logical behavior of the architecture of the system.

The SPES matrix also defines the technical viewpoint that is out of the scope of this work and therefore not presented here.

2.3.5.3 Correctness of a Specification

Correctness of the functional specification B with respect to the requirements specification A is captured by the formula

$$\forall[\wedge B \Rightarrow \wedge A]$$

It expresses that $\wedge B$ is a refinement of requirements specification $\wedge A$.

Analogously, the correctness of the logical architecture C with respect to the functional specification B is described by

$$\forall[\wedge C \Rightarrow \wedge B]$$

2.3.5.4 Consistency of Requirements

A logical view on requirements facilitates a precise definition of some aspects of consistency of requirements. It is assumed that all functional requirements can be represented as logical assertions. Then *logical inconsistency* between logical assertions and sets of assertions can be defined as follows: Two assertions P, Q are *inconsistent* if

$$\neg\exists(P \wedge Q)$$

It may be the case that two requirements are pairwise consistent, but more than two requirements might be inconsistent. As a simple example, the parameter $x \in \mathbb{N}$ shall fulfill all three assertions $(x \leq 5)$, $(5 \leq x \leq 10)$ and $(x \leq 1 \vee 10 \leq x)$. These are pairwise consistent. But $x \in \mathbb{N}$ cannot fulfill $(x \leq 5) \wedge (5 \leq x \leq 10) \wedge (x \leq 1 \vee 10 \leq x)$. Therefore, the definition has to be extended to a set of requirements. A set R of assertions is called *logically consistent*, if the following proposition holds

$$\exists(\wedge R)$$

2.3.5.5 Formal Refinement Specification

In FOCUS, a refinement link between two formal interface specifications can be represented formally as a *formal refinement specification* [Bro10a]. A formal refinement specification consists of a representation and an interpretation function. Based on these functions, the formal refinement specification defines the relations between the input and output ports of the formal representations, see Figure 2.7. The representation function defines how the input ports of an abstract interface specification (for example of a requirement) are mapped to the input ports of a concrete interface specification (for example of a service). Vice versa, the interpretation function defines how the output ports of a concrete interface specification are mapped to the output ports of an abstract interface specification. Formal refinement specifications form the basis for the application of formal verification and test between abstract and concrete specifications.

FOCUS defines three types of refinement:

Behavioral refinement "relates specifications of the same syntactic interface. The refined (more concrete) specification may impose further functional and non-functional requirements in addition to those imposed by the given (more abstract) specification" [BS01, p. 241].

Interface refinement facilitates to document refinements of the interface of a system. It "relates specifications of different syntactic interfaces. The refined specification is a 'behavioral refinement' of the given specification with respect to a translation of its I/O histories" [BS01, p. 241]. For instance, a channel can be refined by two channels, or a message may be refined by several messages.

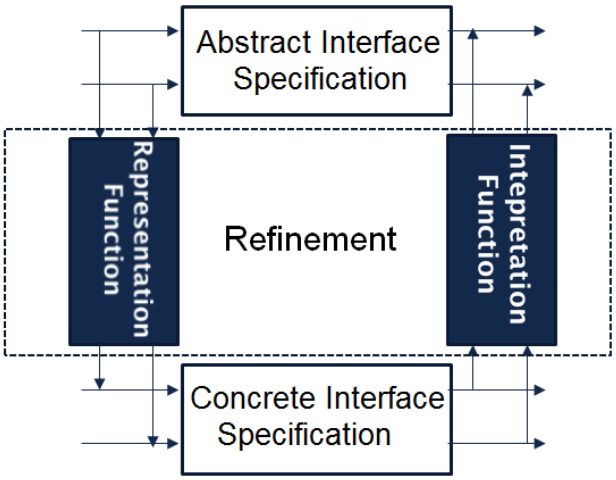

Figure 2.7: A refinement specification specifies how a concrete interface specification is related to an abstract interface specification

Conditional refinement "is a generalization of both behavioral and interface refinement making the strengthening of input assumptions easier and more flexible." [BS01, p. 245]. It "generalizes the relations of behavioral and interface refinement by allowing the introduction of additional input assumptions" [BS01, p. 241]. Conditional refinement includes for example the addition of timing constraints and the replacement of unimplementable data types such as real numbers to implementable data types.

2.3.6 Implementation of FOCUS in AutoFOCUS3

The tool AutoFOCUS3 [HF10, KRSV13, AVT+15, Sch09][1] is a scientific open source tool for the component-based development of reactive, software-intensive, embedded systems. AutoFOCUS3 offers a set of graphical specification techniques and formal quality assurance techniques based on the semantics of FOCUS. AutoFOCUS3 was developed to perform research on tool concepts for model-based development as well as on pragmatic aspects of formal verification and synthesis. While FOCUS permits to define untimed, timed, and time-synchronous specification techniques, AutoFOCUS3 is limited to a time-synchronous notion of streams. A time-synchronous notion of streams is a discrete notion of time based on globally synchronized clocks.

In AutoFOCUS2, the predecessor of AutoFOCUS3, a model-based RE tool (AutoRAID) was implemented, see Section 2.3.7. AutoRAID did not investigate a systematic application of the formal specification and quality assurance techniques of AutoFOCUS3 to the requirements specification. This work investigates this application.

Data Type Definition. AutoFOCUS3 provides a set of elementary data types such as a Boolean data type and a set of natural numbers. In addition, AutoFOCUS3 pro-

[1] http://af3.fortiss.org/

vides a *data dictionary* to define complex data types such as enumerations.

Visual Specification Techniques. Syntactic interfaces $(I \blacktriangleright O)$ and channels between syntactic interfaces are visualized by a *component* diagram with input ports and output ports as depicted in Fig. 2.8.

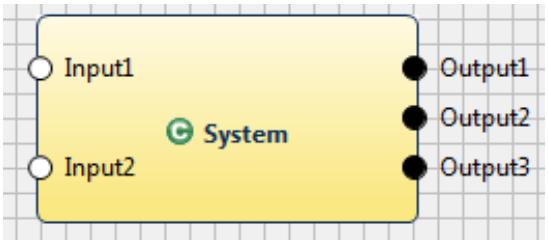

Figure 2.8: The syntactic interface specification of a system as a component diagram with input and output ports

Components can be composed through typed input and output ports which are connected by channels. An example for a component architecture is given in Figure 2.9.

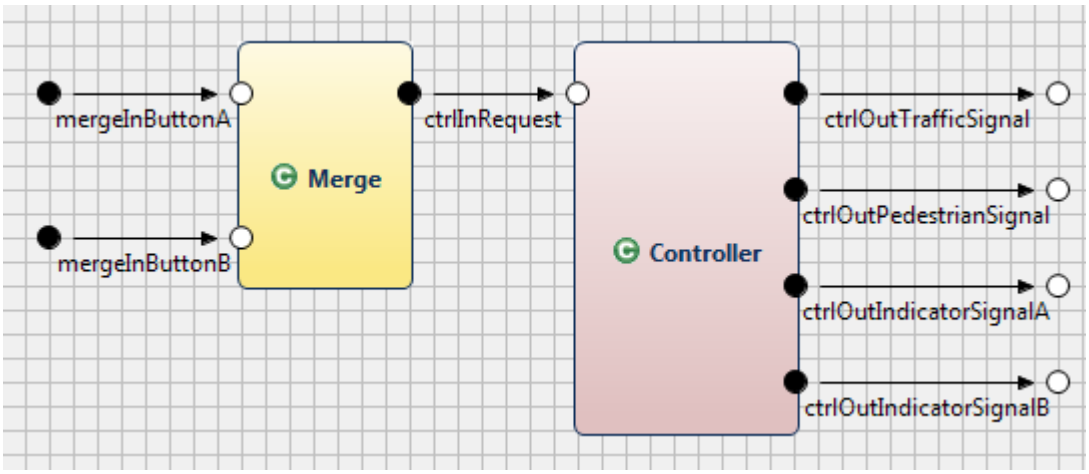

Figure 2.9: Example for a component architecture in AutoFOCUS3

The interface behavior $IF[I \blacktriangleright O]$ of a component can be further detailed:

1. Components can be described by a hierarchy of subcomponents, where the ports of subcomponents are connected with the main component and/or with each other.
2. An atomic component is defined by an interface behavior specification. The interface behavior specification defines the system behavior that needs to be fulfilled by the implementation. If the responses for all potential stimuli are defined, this is called *full behavior*. An incomplete subset of these stimuli and reactions is called *partial behavior*.

To specify interface behavior, AutoFOCUS3 provides the specification techniques *I/O assertions* and *Message Sequence Charts (MSC)* to specify partial behavior and *state automata* or simple imperative code (see Fig. 2.10) to specify full behavior.

```
if (behaviorInState == 0) {
    behaviorOutTrafficSignal = Green( );
    behaviorOutPedestrianSignal = Stop( );
    behaviorOutIndicatorSignal = Off( );
    behaviorOutTime = -1;
    behaviorOutState = 1;
    return;
}
```

Figure 2.10: Code specification in AutoFOCUS3

The following I/O assertions can be specified in AutoFOCUS3. *Assumption/guarantee specifications* define the environment in which an interface specification is executed (see [BS01, p. 213]). *Assumptions* represent invariants over the input ports of an interface. Assumptions describes properties of the environment that have to be satisfied. *Guarantees* represent invariants over the output ports of an interface. Guarantees represent conditions that are required to fulfill by an interface, in an environment where the assumptions are fulfilled. *Basic contracts* represent logical expressions on the inputs and outputs of an interface specification of the form "if assumption A holds then guarantee holds". *Advanced contracts* represent complex conditions on the inputs and outputs of an interface specification. A temporal logics specification consists of one or more contracts (assumptions, guarantees, basic/advanced contracts). All these contracts specify input/output traces of system behavior. The contracts are expressed in temporal logics. AutoFOCUS3 offers *temporal logics specification patterns* as introduced by Dwyer et al. [DAC99] and as depicted in Fig. 2.11. AutoFOCUS3 translates these specification patterns automatically to temporal logic formulas.

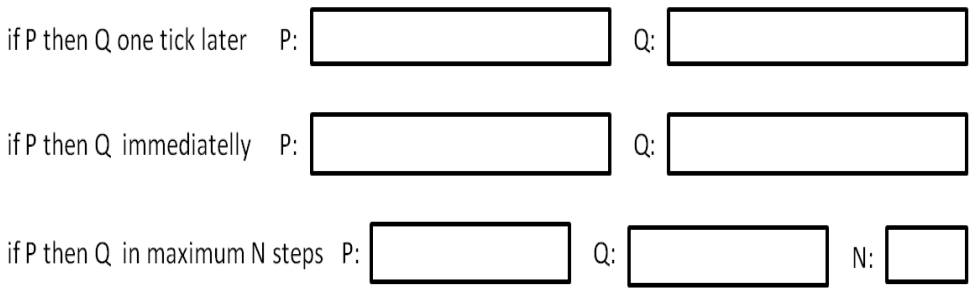

Figure 2.11: Specification pattern for formulas P and Q

AutoFOCUS3 proposes *Message Sequence Charts* (MSC) to formalize the interactions between interface specifications, more precisely, a dialect of the MSC standard [ITU11a] that matches FOCUS. Each MSC consists of the following elements: *MSC entities* define the actors that are interacting. Each MSC entity is an autonomous execution entity and represents an interface specification. The interaction between MSC entities is described by *messages* at defined *entry and exit points*. Entry and exit points denote the syntactic interface of an MSC entity. An example for an MSC is given in Figure 2.12.

A simple input / output *state automaton* model may be used to define stateful interface behavior. The state automaton is described by a set of control states, internal data state variables and state transition functions. One of the states is defined as the

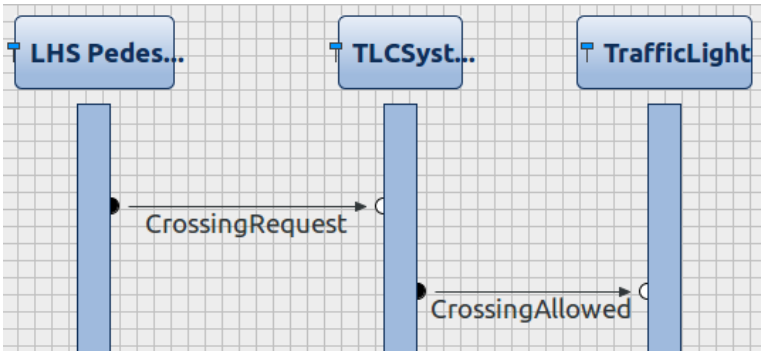

Figure 2.12: Example for an MSC

initial state. Each data state variable has a defined initial value. Each state transition function has a source state and a target state. Each state transition function defines the conditions for a transition from the active source state to a target state depending on the current values of input and data state variables. Furthermore, the transition function can assign values for output variables and data state variables. An example for a state automaton is given in Figure 2.13.

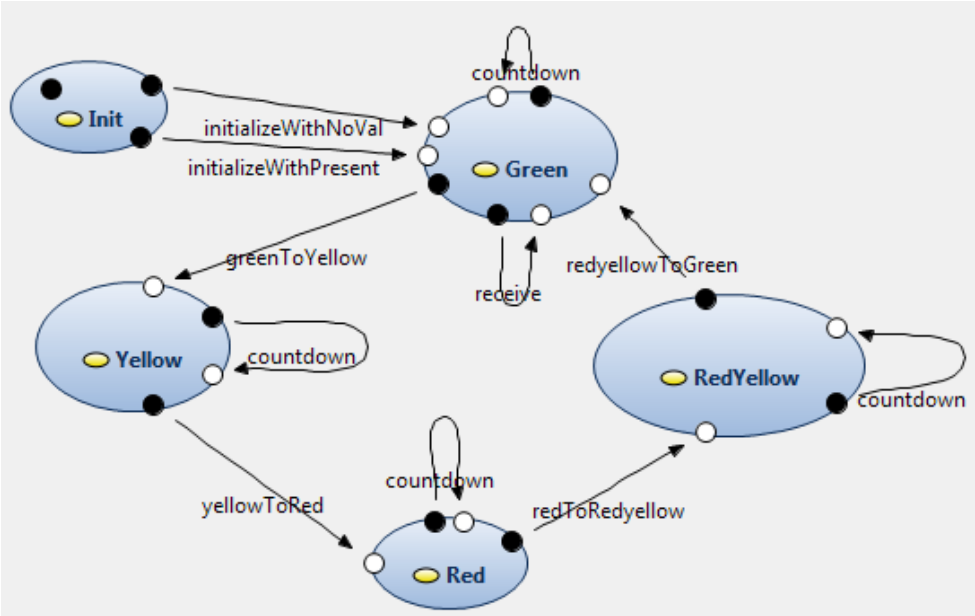

Figure 2.13: Example for a state automaton

AutoFOCUS3 implements *formal refinement specifications* as introduced in Section 2.3.5.5, see also [MR12]. The formal refinement specification can be applied to test cases on executable models on different levels of abstraction [BMR12]. Furthermore, the formal refinement specification can be applied to transform I/O assertions and state automatons.

Modeling the Functional and Logical Viewpoint. In the following, the most important artifacts of SPES and their representation in AutoFOCUS3 are summarized. In AutoFOCUS3, the functional and logical viewpoint of a system under develop-

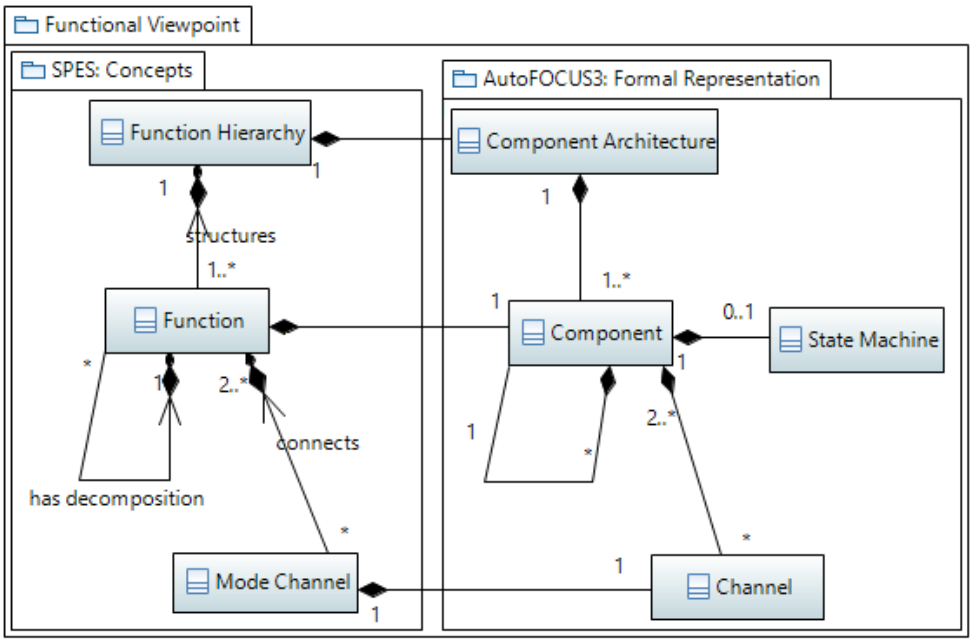

Figure 2.14: Schematic overview of the functional viewpoint

ment can be modeled by a *component architecture*. The component architecture decomposes the system following functional or logical decomposition criteria into atomic components representing functional services or logical components. The behavior of each atomic component can be defined by state automata or simple code as presented above.

The *functional viewpoint* provides a *functional architecture*, a decomposition of a system according to its functions. The functional viewpoint structures the functional requirements for a system under development into *user functions*; these user functions may be decomposed hierarchically into user functions. User functions may be further decomposed into smaller units of functionality that provide an abstract realization of the user function. In AutoFOCUS3, these functions are represented in a *component hierarchy*, where each function can be modeled as a *component*. The hierarchy of the functions is represented in the component hierarchy. If a function is not further decomposed, the *behavior of a function* can be defined; for modeling the behavior, AutoFOCUS3 offers for example *state automata*. Dependencies between functions on the same hierarchical level are modeled as *mode channels*; in AutoFOCUS3, mode channels are represented as *channels between components*. A schematic overview of the artifact reference structure of the functional viewpoint is given in Figure 2.14.

In the *logical viewpoint*, the system under development is decomposed into subsystems. The result is a *system architecture*. The system architecture refines the functional architecture by considering not only functional requirements, but additionally extra-functional requirements and architectural constraints; for example, safety requirements may be realized in redundant subsystems. Therefore, the logical architecture usually differs from the functional architecture. The system architecture structures the system under development in hierarchical logical *components* that represent the system and its subsystems. In AutoFOCUS3, the *behavior of a (sub-)system* is modeled as a *state machine*. *Dependencies* between subsystems are represented as *channels*. A

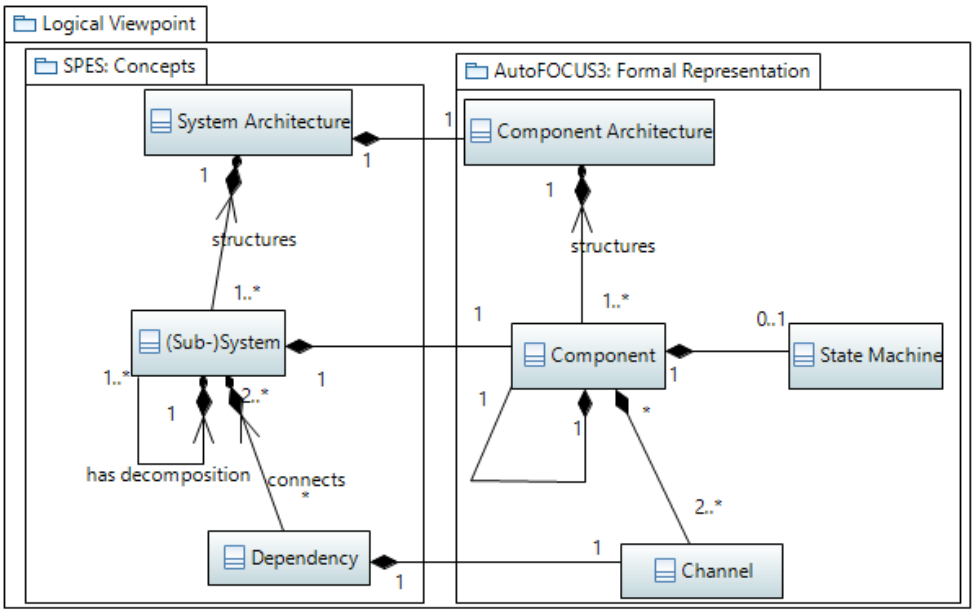

Figure 2.15: Schematic overview of the logical viewpoint

schematic overview of the logical viewpoint is given in Figure 2.15.

Formal Quality Assurance Techniques. AutoFOCUS3 provides a set of formal quality assurance techniques based on the formal specification techniques. Some formal quality assurance techniques require an *executable model*, i.e., a model with execution semantics. In AutoFOCUS3, an executable model refers to the component architecture or parts of it: State machines and code specifications can be used to define the (executable) interface behavior of a component. Components with executable behavior can be hierarchically composed by connecting components via channels.

Simulation executes and animates executable models.

Non-determinism check is performed on a state automaton in order to prove that the state automaton has a deterministic behavior.

Unreachable state check identifies unreachable states in a state automaton.

Assume/guarantee (A/G) reasoning can be applied in two settings: 1) For a component, guarantees on the output ports and (optional) assumptions on its input ports are defined. The component has an implemented interface behavior. A/G reasoning checks that the implemented interface behavior fulfills the guaranteed output values (under the assumed input values). 2) Outputs of component A are connected via channels $c(O_A, I_B)$ with inputs of component B. It can be checked that the guaranteed output values of component A have value ranges that are compatible to the assumed input values of component B.

Formal verification check checks that a component model of the system under development fulfills a set of properties specified as contracts. The component model has to be represented as an executable model. The verification is performed by an automatic transformation of the contract and the executable model to a model-checker. If a counter-example is provided, that counter-

example can be simulated.

MSC conformity check (based on Autili et al. [AIP07]) verifies, whether an MSC is satisfied in a component model. In order to conduct this check, the entry and exit points of each MSC entity of the MSC have to be assigned to the input and output ports to the corresponding components in the component model.

Model-based testing [MR12, BMR12] provides test-suites with specific coverage criteria. The test-suites are generated from an executable model.

These formal analyses are automated by a precise translation from AutoFOCUS3 into the language of the NuSMV[2] [CCGR00] model checker. For details on the integration see Campetelli et al. [CHN11]. The communication of AutoFOCUS3 with the model checker is transparent for the AutoFOCUS3 users. The call and the translations to the model checker and back are done in the background. In cases, in which the model checker found a counter example (for example an infringement of a property), the counter example can be simulated in the AutoFOCUS3 simulator. This thesis investigates how these techniques can be applied in RE.

2.3.7 Requirements Specification with AutoRAID

AutoRAID [SFGP05], [GGS06], [Sch09, p. 21ff] is a model-based approach for RE in the embedded systems domain. The goal of AutoRAID is to avoid the quality issues *inconsistency* and *ambiguity* and to improve *traceability* and increase *verifiability*. AutoRAID was implemented as a plug-in of the predecessor of AutoFOCUS3, in AutoFOCUS2. AutoRAID provides a process for the formalization of requirements from prose to structured text and analysis techniques for structured text requirements. The structured text can be mapped and traced to model elements of the system architecture.

AutoRAID provides an artifact reference structure for requirements that is integrated with subsequent development artifacts. AutoRAID facilitates the documentation of source documents [GGS06]. Requirements are differentiated in *architectural requirements* on the structure of a system, *modal requirements* of the operation modes of a system, *data requirements* of the data communicated or stored in a system, and *functional requirements* on the functions provided by a system. AutoRAID defines attributes for requirements like an ID, title, a patron (the owner of that requirement) and a textual description. Various trace links can be documented: Requirements can be *refined* to more detailed requirements, *associated* with each other or *motivate* architectural decisions. Requirements are differentiated according to their impact on the architectural model; each requirements type *motivates* a specific model element in the architecture model of the system under development. For example, architectural requirements motivate components in a component architecture describing the logical viewpoint of a system.

AutoRAID [Sch09, p. 21ff] proposes a guideline for the review-based step-wise formalization from requirements written in prose to requirements that are structured and detailed by attributes. The approach supports the specification, tracing and analysis of textual requirements by providing an artifact reference structure of the

[2] http://nusmv.fbk.eu/NuSMV/

requirements specification. It traces the attributed requirements to a formal design model. The approach comprises the following actions:

1. Individual requirements are identified by breaking down the requirements specification into coherent parts. These requirements are iteratively broken down into sub-requirements until the requirements can be classified with respect to the different requirement types defined by AutoRAID.
2. The break-down relationship of requirements is documented as refinement links. Furthermore, requirements addressing a common issue can be traced.
3. Requirements are classified into four different types (architectural, modal, data, and functional requirements). For each requirement type, specific attributes are defined that have to be documented.
4. The attributes of the requirements are associated with model elements of a design model by specifying a *motivated-by* association.
5. Each requirement type motivates specific model elements of the design model. In the example of a typed port, an architectural requirement motivates the port; a data requirement motivates the data type. Both requirements are integrated in the design model.
6. Automated conformance checks indicate whether the resulting requirements specification conforms to rules defined on the AutoRAID artifact reference structure. An example for such a conformance check is "Each requirement must be classified or refined by a further requirement" [Sch09, p. 45].

Inconsistencies are successively removed by successively breaking down requirements into a detailed model with linked elements. Ambiguity is addressed by using a detailed artifact reference structure, expressing the requirements in terms of this model successively leads to elimination of ambiguities, for example, by classifying and detailing the steps of a scenario. Linking requirements to elements of the design ensures traceability is automatically. Through the restricted expressiveness for requirements, the description of a requirement is design-like, thereby reducing the risk of infeasibility and increasing verifiability.

AutoRAID does not provide a formal representation of requirements that is independent of the system architecture. Furthermore, AutoRAID does not specifically support the validation of functional requirements.

2.3.8 Requirements Formalization and Analysis for Hybrid Systems

Cimatti et al. [CRST09, CRST13] have developed a model-based RE approach for hybrid systems, where discrete and continuous components are tightly intertwined. It includes a modeling language called OTHELLO for the requirements specification. The expressiveness of OTHELLO has been tailored to represent functional requirements. Based on the modeling language, a guideline directs the formalization and analysis of requirements.

OTHELLO is a self-developed visual formal modeling language adopted from UML and enriched with a subset of the property specification language PSL [Eis07]. OTHELLO defines a set of RE concepts and their formal representation. OTHELLO fa-

cilitates a set of formal analysis techniques. The approach provides methods for consistency checks (a set of behavior constraints under given value constraints are satisfiable), scenario compatibility check (a set of scenarios is compatible with a set of behavior and value constraints) and property checks (a property is implied by a set of behavior and value constraints).

Cimatti et al. propose conducting three steps for the specification and quality assurance (QA) of requirements. The approach assumes that the requirements specification is documented as unstructured text. From this starting point, three main actions are recommended:

1. *Informal Analysis.* The requirements specification is split into requirement fragments. requirement fragments are *classified* into different RE concepts (glossary, architecture, functional, communication, behavioral, environmental, scenario, property, annotation). The approach provides a question for each type that leads the categorization. Between the classified requirement fragments, links can be defined: strong dependency link ("A cannot exist without B"), weak dependency link ("A can exist without B"), refinement link ("A redefines some notions of B at a lower level of abstraction"). An inspection identifies flaws in the informal requirement fragments.
2. *Formalization.* The fragments are *formalized* following predefined target notations. For each RE concept, Cimatti et al. propose a specific modeling notation. For example, functional requirements should be formalized to state automata. Glossaries should be formalized to classes and class diagrams. The resulting models are either directly derived from a requirements fragment or include several requirement fragments. The requirements fragments are linked to the resulting model elements.
3. *Formal Analysis.* The resulting formal models are *checked syntactically*. Then, an analysis expert *selects* a subset of the formalized requirement fragments and selects automatic checks in order to identify quality issues in the requirements. The selected subset is analyzed to identify possible flaws.

The approach provides a modeling language and guidance for the formalization and the formal analysis of functional requirements in order to detect quality issues. The approach does not address the validation and verification of functional requirements. Operationalization is only provided by setting up a tool-chain [CRST13]. This thesis integrates the specification and formal analysis activities proposed in the guideline.

Chapter 3

Related Model-based RE Approaches

This chapter presents those works that are related to MIRA, the model-based RE approach developed in this thesis. Work is related, if it presents a model-based RE approach that addresses the four challenges stated in Chapter 1 and if it addresses functional requirements. A vast number of literature on model-based RE approaches has been published since the beginnings of model-based RE in the late 1970s. We chose a systematic approach to investigate these publications. We defined two sets of criteria to characterize related work. The first set serves to identify related work. The second set is used to discuss related work. We applied a Systematic Mapping Study (SMS) [PFMM08] to identify and structure the literature on model-based RE according to the first set of criteria. From this pool of work, we filtered the papers that match the characteristics for related work. We identified further related work that was outside the scope of the SMS. The related work is discussed in detail based on the second set of characteristics.

This chapter is based on an SMS performed by Ezaga [Eza15] in the course of his Master thesis. The Master thesis was supervised by the author.

Contents

3.1 Research Method

The goal of the study is to identify the works that are related to the MIRA approach developed in this thesis. We identified related work using an SMS, a research methodology that reviews a number of works in order to structure a research area. The SMS was conducted by adapting the guidelines provided by Peterson et al. [PFMM08]. A search process identified papers about model-based RE. We then defined the characteristics of related work. We elaborated a scheme to categorize the

identified papers into those that meet the characteristics and those that do not. We applied this scheme on the papers identified in the search. The result of the SMS is a categorized list of papers. We down-selected the related work from this list. Finally, we assessed the related work regarding detailed characteristics for a comparison with MIRA.

3.1.1 Search Process

The search process has the goal to identify work in the field of model-based RE. In the following, we describe the inclusion and exclusion criteria we applied for the identification and the search databases we used.

Inclusion and Exclusion Criteria. In this study, we considered approaches that focus on the requirements for a system under development and not on other requirements, for example, requirements for the development process. In addition, approaches should use models to provide a) specification techniques to represent requirements, b) a system modeling theory to interpret requirements, or c) model(s) that define the contents of a requirements specification and their representation (artifact reference structures).

The search process was guided by inclusion and exclusion criteria. A publication was included when it:

- applied, proposed, evaluated, discussed, opined, reviewed and/or envisaged for the future, novel or existing models (or model-based approaches) on RE;
- uses models that match the properties stated above;
- is either a paper from a journal, conference or international symposium;
- contains enough information to be classified according the criteria for related work;
- was published up to 2014.

A publication was excluded when it

- did not explicitly address RE;
- is not from a journal, conference or international symposium;
- did not provide access to full text of research work for some reason;
- was not written in English language.

Search Databases and Search Strings. We searched five online publication databases, the IEEE Explore Digital Library, the ACM Digital Library, Springer Link, ScienceDirect and Google Scholar. Each of the five databases was searched with a search string. For IEEE and Science Direct the same search string was used. Google Scholar had minor modifications of this search string while ACM and Springer had different variations of search string as show in Section 3.1.1. The search string for IEEE, Springer and ACM databases resulted in around 2000 papers. Google Scholar and Science Direct provided access to around 1000 publications each. During the

Table 3.1: Search databases and search strings

Database	Search String
IEEE, Science Direct, Google Scholar	((Requirement engineering) AND (*based OR *oriented OR model-driven OR content model OR concept model OR formal specification OR requirement structuring OR requirement classification OR formal approach OR mathematical model OR high level abstract model))
ACM	(requirements engineering) and (Keywords:requirements/specifications, OR Keywords:requirements/analysis, OR Keywords:requirement/framework, OR Keywords:model/based, OR Keywords:model/driven/requirements, OR Keywords:content/model, OR Keywords:concept/model, OR Keywords:requirement/development, OR Keywords:requirement/modeling, OR Keywords:formal/requirement/ model)
Springer Link	requirement AND engineering AND (requirements OR model-based OR approach OR model-driven OR concept OR content OR model OR formal OR specification OR classification OR mathematical OR system OR development)

search and study selection process, a researcher screened a total of 8000 papers in their titles and abstracts, applying the inclusion and exclusion criteria. Where insufficient information was given to judge on the inclusion and exclusion criteria, the conclusion (and in some cases the entire publication) was read before deciding to include or exclude papers. A second researcher, the author, reviewed these papers and performed manual search to add more papers. The search resulted in a final set of 336 publications.

3.1.2 Definition of the Characteristics of Related Work

The scope of this thesis infers the initial characteristics for related work: Firstly, related work presents a model-based RE approach. Secondly, the model-based RE approach handles functional requirements.

The first challenge is to support heterogenous requirements specifications. Related work should facilitate the representation of both textual and formalized requirements.

The second challenge is to cover the analysis, validation and verification of requirements. As names for the different quality assurance activities differ in literature, related work should facilitate any quality assurance of requirements. We then assess related work in more detail for the various quality assurance activities to see, whether the approach covers the analysis, validation and the verification of functional requirements.

The third challenge is to be integrated in a seamless model-based development approach. A necessary precondition for related work is to be classified as formal, since

seamless development requires defined semantics. We then assess in detail whether the related work is integrated in a seamless model-based development approach.

The fourth challenge is guidance and tool support. Hence, we investigate the related work in more detail according to these two characteristics.

3.1.3 Identify Related Work

We identify related work using the characteristics discussed in Section 3.1.2. Summarizing, related works have the following characteristics:

- The work presents a model-based RE approach.
- The model-based RE approach handles functional requirements.
- The model-based RE approach facilitates the representation of both textual and formalized (heterogenous) requirements.
- The model-based RE approach facilitates a quality assurance of requirements.
- The model-based RE approach is formal.

We proposed a classification for each of these characteristics (see below). We mapped the 336 publications to these classifications. This provided quantitative insights into the characteristics. Using the classification, we then down-selected the publications to the related work. After down-selecting the papers to the related work, we applied snowball sampling [BW81] to identify additional related work. In addition, we identified and added related work that was published after 2014. Furthermore, we extracted tools that were mentioned in the papers.

Criteria for a Model-based RE Approach. A publication can be classified according to the seven different research type facets, six of which are described by Wieringa et al. [WMMR05]:

Solution proposals present one or more techniques with descriptions and rationale, as a solution to a problem.

Validation research papers are a subset of solution proposals, which are not yet implemented in practice. They would have been described and explained with illustrative examples.

Evaluation research papers present solution techniques which have been evaluated in practice. Solutions which have been tested in an academic setting such as a university with students or in an industrial setting qualify as those evaluated in practice. In addition, non-solution proposals which mainly review and evaluate some existing technology also qualify as evaluation research papers.

Philosophical papers are unique in that they introduce some new paradigm or conceptual framework for structuring investigative activities, for example, a new way of looking at requirements.

Opinion papers present the authors' opinions and views concerning how certain issues should be approached, the true meaning of concepts or reveal their viewpoint about past events such as reasons behind the software crisis.

Experience papers are useful in presenting the experiences of practitioners in trying out a particular technique.

Vision (or Road-map) papers undertake a review of current state of research and then propose one or more directions to which it should be focused in the future.

We classified the identified papers according to this scheme. Overlaps were taken into account, i.e. there were papers which qualified into more than one facet, to a maximum of three facets. Related work should present an *approach*. Hence, we selected only solution proposals, validation and evaluation research and experience papers as related work.

Criterion for Functional Requirements. We extracted the requirements type that an approach addresses from each of the papers. We classified these types as functional and extra-functional. Based on this information, we could assign each paper to the group of approaches that handles functional requirements, extra-functional requirements or both. We assigned work to both, if it explicitly was applicable to both or if the approach did not limit its applicability to a specific type of requirement. We selected approaches for functional requirements and for both as related work.

Criterion for Heterogenous Requirements. We investigated whether contributions facilitate the conversion of textual requirements into a semi-formal or formal representation. The conversion can be automatic, semi-automatic or manual, then provided with explanations for the conversion. Approaches could be classified as heterogenous or not heterogenous. Related work comprises heterogenous approaches.

Criterion for Quality Assurance. In a bottom-up-approach, we extracted the purposes from the papers. A purpose indicates an activity or RE phase that a model-based approach aims to support. These purposes have been grouped into quality assurance and other purposes to identify related work. Related work targets quality assurance, but additionally may also target other purposes.

Criterion for a Formal Approach. We extracted the modeling languages that the approaches use and classified them as formal or semi-formal according to our definition of these terms and using further information in the papers. We identified three categories: Semi-formal approaches use semi-formal languages, formal approaches use formal languages, hybrid approaches combine semi-formal and formal modeling languages. Related work consists of approaches that are formal or hybrid.

3.1.4 Discussion of Related Work

According to Section 3.1.2, we discussed the related work in detail with respect to these characteristics:

- The model-based RE approach covers the analysis, validation and verification of functional requirements.

- The model-based RE approach is integrated in a seamless model-based development approach.
- The model-based RE approach provides a method and tool support.

Related work that covers these criteria addresses the same challenges as MIRA.

3.2 Study Results

The SMS identified 336 original research papers in the field of model-based RE from 1976 until 2014. Details on the case study and its results can be found in Ezaga [Eza15]. We limit the presentation of study results to the classification and down-selection of related work. We provide some examples for excluded work that are either well-known (according to the opinion of the author) or that match some of the criteria. Then, we discuss related work in detail in Section 3.3.

272 papers presented solution papers. Another 36 papers present validations, evaluations and experiences of and with model-based RE approaches. Hence, in total 308 papers are subject to further investigation.

199 papers handle functional requirements, and further 50 handle functional and extra-functional requirements. The remaining papers handle only extra-functional requirements.

Only 37 papers present approaches that include both textual and semi-formal/formal requirement models. The down-selection excludes those papers which start with modeling or whose inputs are models. This criterion excludes approaches such as the Software Cost Reduction (SCR) approach [HJL96, HJ07], the Analysis and Description of Requirements and Architecture (ADORA) approach[1] [GBJ02], RAT [BCP+07], and RATSY [BCG+10]. This criterion also excludes the commercial tool STIMULUS[2] [GJ16] that provides capabilities to model and simulate functional real-time system requirements and test scenarios.

208 papers present formal or hybrid approaches. Semi-formal approaches that do not provide an underlying formal system model are excluded. Examples for semi-formal modeling languages are the Unified Modeling Language (UML) [OMG11] and the System Modeling Language (SysML) [OMG12]. UML was the most predominant modeling language in the SMS with over 100 publications. This criterion also excludes UML- and SysML-based approaches such as Rational Unified Process (RUP) [Jac92, Sof11] or COMPASS (Comprehensive Modeling for Systems of Systems) [HPP+15]. Included are hybrid approaches that tailor UML to a formal modeling language. For example, fUML [Gro13] extends a subset of UML to be executable. Romero et al. [RSF14] use fUML for a formal analysis applying theorem proofing.

218 papers claim to address quality assurance. The other approaches state purposes like negotiation, code generation, or traceability or do not explicitly name a specific purpose other than specification.

Taking into account all these characteristics at once, the SMS identified 24 papers

[1] http://www.ifi.uzh.ch/rerg/research/adora.html, last accessed 10-29-2016
[2] http://argosim.com, last accessed 10-29-2016

that constitute related work. Snowball sampling and an investigation of work newer than 2014 revealed additional related work. In total, we identified 29 approaches described in 39 papers that constitute related work.

3.3 Discussion of Related Work

In the following, we discuss the related work with respect to the four challenges stated in Chapter 1. Table 3.2 summarizes the extent to which each of the discussed approaches addresses each challenge.

3.3.1 Challenge 1: Heterogenous Requirements Specifications

Many approaches such as Rockwell Collins [MTWH06, Hei07] provide a means to represent requirements using natural language and a formal language. A variety of approaches such as Cimatti et al. [CRST13] and KAOS [vL01] do not limit heterogeneity to requirements, but even provide other contents of the requirements specification in different representation forms. Two approaches [KJL09], [SBC10] limit heterogeneity to domain knowledge. Therefore, these two approaches do not fully address the first challenge.

3.3.2 Challenge 2: Analysis, Validation and Verification of Functional Requirements

We investigated whether an approach covers the analysis, validation and verification of functional requirements. A variety of approaches are limited to explicitly support one or two of these quality assurance activity, whereas MIRA comprises all three activities. We present the approaches grouped by the number of activities they support.

One Quality Assurance Activity. A variety of model-based RE approaches provide support for the analysis [LCK98, JMM99, CRST13, Ili07, dSAVP10, KP11, KC05b, KJL09, FMK+11, GBC+07, FdS12, NEA12], validation [HD98, PDC+11], or verification [SJV12, JHLR10, BCMW15] of functional requirements. In the following, we present some of these approaches in more detail.

Some earlier approaches developed methods to perform a quality assurance activity on a specific type of requirement. Heymans and Dubois [HD98] investigate the formalization of scenarios to a formal, MSC-like notation and its validation through simulation. Lee et al. [LCK98] formalize use cases to petri nets to then analyze them.

The approach by Cimatti et al. [CRST13] was introduced in Chapter 2. It is designed for hybrid systems and therefore could be applied for embedded systems. The approach uses the OTHELLO language. It proposes a method for the formalization and the analysis of requirements and other contents of the requirements specification (e.g., glossary) to detecting flaws in the requirements specification. The Cimatti approach does not cover the validation or verification of requirements.

Table 3.2: Summary of related approaches with respect to the challenges (C.1) heterogeneity, (C.2) coverage of analysis, validation and verification, (C.3) seamless and (C.4) guidance and tool support. The table lists the name or the first author of the approach, and the papers in which the approach was presented. "+" means that the approach fully addresses that challenge. "o" indicates that the approach addresses some aspects of the challenge, but does not completely cover it. The last row of the table provides the sum of challenges that were fully addressed

Name / Author	Paper(s)	C.1	C.2	C.3	C.4	Sum
Kroha	[KJL09]	o	o	o	o	0
Sanyal	[SBC10]	o	o	o	o	0
SOLIMVA	[SJV12]	+	o	o	o	1
Jastram	[JHLR10]	+	o	o	o	1
Backes	[BCMW15]	+	o	o	o	1
HiLiTE	[BS10, BBB+12]	+	o	o	o	1
Heymans	[HD98]	+	o	o	o	1
Pires	[PDC+11]	+	o	o	o	1
Lee	[LCK98]	+	o	o	o	1
Juristo	[JMM99]	+	o	o	o	1
Cimatti	[CRST13]	+	o	o	o	1
Ilic	[Ili07]	+	o	o	o	1
De Sousa	[dSAVP10]	+	o	o	o	1
Kof	[Kof10, KP11]	+	o	o	o	1
Njonko	[NEA12]	+	o	o	o	1
SPS	[KC05b, PMHP12]	+	o	o	o	1
DODT	[FMK+11, SW14]	+	o	o	o	1
Gorse	[GBC+07]	+	o	o	o	1
RSL-IL	[FdS12]	+	o	o	o	1
AutoRAID	[SFGP05, GGS06, Sch09]	+	o	o	o	1
BTC	[Jus13, ESH14]	+	o	o	o	1
Kim	[KS04]	+	o	o	o	1
Sengupta	[SD15]	+	o	o	o	1
General Motors	[RG11]	+	+	o	o	2
Rash	[RHR+06]	+	o	+	o	2
FOCUS	[BS01]	+	o	+	o	2
ARIES	[JFH91]	+	+	+	o	3
KAOS	[vL01, LvL02, vL03, DDR03]	+	+	+	+	4
Rockwell Collins	[MTWH06, Hei07]	+	+	+	+	4

Ilic [Ili07] introduces an approach to formalize requirements to the B Method. To support the transformation, he proposes requirements templates with structured text for events, data and timing requirements. Requirements are classified as events, data

or timing requirements. The templates are mapped to corresponding B language constructs. For example, a requirement in an event template is formalized to an event in B. The field determines which information becomes which part in the B event specification. This approach is very similar to the formalization proposed in this thesis. The publication does not discuss the validation or verification of requirements or investigates tool support. De Sousa et al. [dSAVP10] formalize use cases to the B method and perform consistency analysis.

Several approaches investigate the automation of the formalization of natural language requirements to formal specifications, for example, from scenarios to message sequence charts [KP11]. Njonko and el Abed [NEA12] proposed an approach for the automated formalization from natural language requirements to executable models. The formalization reveals inconsistencies and incompleteness in requirements. Another goal of the formalization could be to generate test-cases from the formal representation of requirements [SJV12]. Konrad and Cheng proposed a specification pattern system (SPS) [KC05b] for controlled English grammar that can be transformed to logics automatically. A case study [PMHP12] applied SPS to automotive requirements of BOSCH in order to analyze sets of requirements for errors. The case study formalized 289 requirements and indicated that SPS is applicable for the formalization when adding three further patterns. The paper did not report on concrete benefits or usefulness of SPS for the requirements analysis.

Jastram [JHLR10] uses the WRSPM reference model [GGJZ00] as a basis for the formalization from natural language requirements to specifications in Event B. This could be a good starting point for a systematic verification of a system against its requirements. Nonetheless, the approach as presented in the paper only provides details on how to verify formal requirements against their textual representation.

Kroha et al. [KJL09] propose the use of ontologies for consistency checking. Ontologies are also applied in the DODT tool [FMK+11] for the requirements analysis.

Pires et al. [PDC+11] combine controlled natural language with UML and ontologies to promote the understanding of requirements amongst stakeholders.

Backes et al. [BCMW15] provide an approach to formalize and verify requirements using compositional verification. They extended the Architecture Analysis and Design Language (AADL) [FG12] that was designed for embedded real-time distributed systems. The approach has been applied on a Quad-redundant Flight Control System within NASA's Transport Class Model. The validation and analysis of requirements is not addressed in this approach.

Two Quality Assurance Activities. Some approaches provide support for the analysis and validation [KS04, SBC10, SD15].

Kim and Sheldon [KS04] present a method to analyze and validate requirements using two formal representations. The formalization identified incomplete and inconsistent requirements. The authors apply fault-injection to obtain a more fault-tolerant system. Sanyal et al. [SBC10] propose an automated method to formalize a domain model from textual requirements. This domain model facilitates the analysis and validation of the requirements. Sengupta and Dasgupta [SD15] propose a method that combines formal and semiformal techniques to model software requirements

for analysis and validation.

Three Quality Assurance Activities. Some approaches [JFH91, vL01, RG11, MTWH06] promise to cover all three activities.

ARIES [JFH91] envisions an approach and tool support for the analysis, validation and verification of the requirements specification. The core idea is that the requirements phase defines textual requirements that are integrated in a formal requirements model. This formal model is then mechanically transformed into an optimized program. We could not identify a follow-up publication that fully demonstrates these capabilities.

KAOS [vL01] is a modeling language that facilitates to semi-formally model goals, their refinement to requirements, operations and objects. KAOS facilitates the description of entities as prose and a formal definition in temporal logic. KAOS supports the elicitation, validation and analysis of requirements. KAOS provides a method to derive operational software specifications from a goal model [LvL02, vL03].

General Motors developed a model-based approach for the formalization from requirements specified in controlled natural language to a formal representation that is then analyzed and verified by simulation, model checking and generated test cases [RG11]. The publication presents test case generation supported by a tool called ReMOTGen. The publication does not provide details on the concrete formalization process and how to perform the transition from requirements to design, or on other quality assurance techniques than test case generation, so we cannot evaluate this approach in detail.

Rockwell Collins report on successful experiments with the application of formal methods in RE in the avionics and aerospace domain and its benefits for verification and certification, especially in the context of domain-specific certification standards. In an industrial context, Rockwell Collins [MTWH06] report on the successful application of the formal analysis and verification of requirements for a flight guidance system. Natural language requirements for a flight guidance system are written in 'shall' form using a formal specification language. The formalization, the formal analysis and the formal verification revealed errors in the natural language requirements and the formal models. The case study demonstrates the applicability and effectiveness of formal analysis tools to real systems. Based on that case study, Heimdahl [Hei07] describes the formalization process and current challenges for safety-critical systems.

3.3.3 Challenge 3: Seamless Model-based Development

Generally, it cannot be assumed that requirements are expressed with the same vocabulary and the same level of abstraction as the system design. An integration of the model-based RE approach in a seamless model-based development approach facilitates a coordinated transition from requirements to subsequent design models.

Some approaches such as Backes et al. [BCMW15] or Justice [Jus13] assume that requirements are expressed at the same level of abstraction as the system architecture.

For example, in the approach presented by Backes et al. [BCMW15], the authors can apply formalized requirements directly to verify system components. This suggests that the requirements might contain architectural decisions that have been undertaken to obtain the architectural model of the system under development and are therefore not independent of these design decisions. Similarly, AutoRAID [SFGP05] avoids an explicit refinement from requirements to the design. Requirements are formalized as part of a system model at the level of abstraction of the system design.

Santiago Junior et al. [SJV12] leave the transformation of abstract test cases on the requirements level to executable test cases to the user of their approach. The refinement from the level of abstraction of the requirements to the system architecture remains implicit and undocumented.

Some approaches such as ARIES [JFH91], KAOS [LvL02, vL03] and Rockwell Collins [MTWH06] propose the transformation of requirement models to subsequent design models of the software under development by pre-defined rules or transformation patterns. Ideally, these models are then correct by construction and do not require any verification. These pre-defined transformation rules need to include all potential design decisions that have to be undertaken during the system development. Automated model transformation seems feasible for narrow, well-understood domains, but the modeling languages require significant customization before the languages can be applied in practice [WHR14]. Therefore, this thesis takes a different approach. We do not prescribe whether subsequent design models are created automatically. We do allow for design decisions that are not encoded in rules. Therefore, a verification of the subsequent design model against its requirements is necessary and supported by the approach.

The formal modeling theory FOCUS provides a solution to close the gap between functional requirements and subsequent development artifacts. FOCUS defines a formal refinement specification that documents how system inputs and outputs are refined over different levels of abstractions. Formal refinement specifications are for example applied in model-based testing [MR12]. In test-driven development, test cases are generated from executable requirements. These test cases can be automatically converted and run a model of the system architecture or code. In order to address seamless model-based development, MIRA integrates the system modeling theory FOCUS.

3.3.4 Challenge 4: Guidance and Tool Support

As discussed above, most approaches are generally limited with respect to quality assurance. Only few approaches address the analysis, validation and verification of requirements. Therefore, guidance and tool support for these approaches is also limited to a subset of the quality assurance activities. Tool support includes a range of research prototypes [RHR+06, KP11, HD98, KC05a] and more mature tools that we will present in the following.

Guidance for analysis, validation and verification is given in two approaches, in KAOS [vL01, LvL02, vL03, DDR03] and by Rockwell Collins [MTWH06, Hei07]. KAOS is supported by a dedicated tool, the Objectiver tool[3]. FAUST [DDR03] is

[3] www.objectiver.com, last accessed 8-31-2015

an extension to the Objectiver tool that facilitates a formal analysis of requirements. Rockwell Collins used a tool chain with a commercial RE tool as a front end.

The DODT tool [FMK+11] supports to formalize requirements using controlled natural language and domain knowledge using ontologies. The domain ontology in DODT includes a thesaurus, facilitates the documentation of relations between terms in the thesaurus and enables interference checks. DODT also provides a set of analyses on requirements expressed by controlled natural language and the domain ontology, including a check for missing requirements and inconsistent requirements. DODT was successfully applied to improve the quality of requirements by reducing ambiguities and inconsistent use of terminology, removing redundant requirements, and improving partial and unclear requirements [SW14]. DODT does not specifically support the validation of requirements by stakeholders or the verification of the system under development against the requirements.

The Honeywell Integrated Lifecycle Tools & Environment (HiLiTE) [BS10] is an in-house tool of Honeywell for the requirements-based verification of aerospace system components that are designed using MATLAB Simulink/Stateflow [ABR05] models. HiLiTE has been applied in the analysis and verification of control systems of several commercial avionics production programs. Using HiLiTE, Honeywell report on a reduction of the cost and time of certain component-level verification tasks that are required by the domain-specific safety standard by a factor of 20 - 50 compared to traditional methods [BS10]. The analysis and validation of requirements is not in scope of HiLiTE.

The BTC EmbeddedSpecifier[4] [Jus13] is a tool that supports creating and managing semi-formal and formal requirements. Starting with informal textual requirements, a formal and machine-readable specification can be derived step-by-step. This specification is at the same level of abstraction as the architecture models [Jus13, p. 11] and can be used for an automated formal verification. Ellen et al. [ESH14] developed an extension for the analysis of requirements, where requirements are formalized independently of the design. The BTC EmbeddedSpecifier differs from MIRA as it does not provide a notion of refinement. Furthermore, the validation of requirements is not in scope of the BTC EmbeddedSpecifier.

Current dedicated commercial RE tools belong mostly to the group of modeling tools or to the group of requirements management tools. Requirements management tools such as IBM Rational DOORS[5] do not specifically support model-based RE. Requirements management tools typically concentrate on the documentation, management and tracing of requirements. In the example of DOORS, a formal representation of requirements is not explicitly supported. Models can be integrated via the Rational DOORS Analyst Add On. This tool allows supplementing textual requirements with pictures for the visualization and simulation of complex systems based on UML and SysML. Rockwell Collins [MTWH06] used DOORS for the formal specification of requirements by defining a separate column for the formal representation of a requirement.

Several commercial modeling tools such as the Enterprise Architect[6] or Papyrus

[4] https://www.btc-es.de/index.php?idcatside=52&lang=2, last accessed 5-3-2016
[5] http://www-03.ibm.com/software/products/en/ratidoor, last accessed 8-31-2015
[6] http://www.sparxsystems.de/uml/ea-function/, last accessed 5-3-2016

[GDTS07] are available. Papyrus [GDTS07] is a modeling tool that can be combined with Moka[7] to define and execute fUML models. These two tools support a range of modeling languages such as UML and SysML. Both tools offer a broad range of features related to modeling, but lack of specific methodical support for model-based RE. Nevertheless, UML/SysML and the corresponding modeling tools typically can be configured and extended in many aspects. Therefore, they should be able to support artifact-orientation and seamless model-based development by customization. First discussions with a tool vendor confirmed this, but revealed that the adaptation would definitely require significant additional efforts. For example, for the operationalization of their model-based RE approach, Cimatti [CRST13] combined the requirements management tool IBM Rational RequisitePro, Microsoft Word and UML modeling tool IBM Rational Software Architect with an extended version of the NuSMV model checker [CCGR00].

3.4 Threats to Validity and Limitations

In the following, we discuss the main threats to validity and limitations of this literature study.

Construct Validity. Construct validity concerns establishing assurance that the study design reflects the objectives of the study. A crucial point is the search string used to identify the primary studies. Different search strings may yield fundamentally different results. We identified and investigated potential search keywords in several iterations. Potential synonyms were extracted from related publications and books of this field. Finally, the search string was fine-tuned according to the capabilities of the search engines. Nonetheless, the list of resulting papers might be incomplete and additional search terms might increase the list of resulting papers. In order to maintain a high quality of the publications, only journal articles and main conference papers have been included to the study. Workshop papers have been excluded.

The search databases also influence the number and quality of papers. The chosen databases include renowned RE journals and conferences, for example, the Requirements Engineering Journal published by Springer, or the IEEE Requirements Engineering Conference.

Some publications extend existing approaches and elaborate certain aspects of an approach. These publications do not provide a coherent overview. This means that where a complete approach may exist, it could be spread over several publications. Hence, some of these approaches may offer solutions to all four challenges. Nonetheless, if each of the publications only covers certain aspects of an approach, it remains unclear whether these aspects can be put together effectively. In order to be able to evaluate such an approach, a publication still needs to summarize the various solutions in a coherent approach.

Internal Validity. Internal validity deals with the data extraction and the data analysis. The paper identification and the mapping of the papers was carried out by

[7] https://wiki.eclipse.org/Papyrus/UserGuide/ModelExecution

one researcher. This means that researcher bias could influence the study results. Nonetheless, the results of each step of the study have been discussed with a second researcher, especially those where doubts in the inclusion/exclusion criteria or the mapping existed.

The mapping from papers to the research questions could also result in misclassifications. This threat was partly addressed by using a widely accepted classification scheme on research facets [WMMR05]. For other classifications, we relied on the terms used by the authors, and avoided misclassifications by aggregating terms with contradicting usage. Another threat to validity stems from the broad scope of the study. The resulting high number of results makes the evaluation an elongated task, which also may lead to misclassifications.

In order to mitigate researcher bias, an iterative approach was taken to minimize misclassifications. The papers have been evaluated several times to reduce researcher bias. A second researcher (the author) controlled the results in random samples on 10% of the papers, thereby identifying only few (less than three) changes.

External Validity. External validity is concerned with the generalizability of the results. The SMS was performed on a 38-year horizon. This means that the SMS is representative in terms of time for model-based RE. The search string comprises relevant terms related to model-based RE and the number of identified papers is extensive with 336 papers. The complete list of all the papers incorporated in this study is provided online [8]. This list and the classification scheme can be used to extend the study with additional papers in the future.

Limitations. Some approaches are spread over various publications, which makes it difficult to identify them as related work. Furthermore, the analysis, validation and verification activities are named completely differently by different authors. For example, some authors call the identification of problems within the requirements specification analysis (like in this thesis) while others call it verification or even check. This increases the difficulty of a comparison. Finally, some publications that seem relevant do not give enough details for an in-depth comparison. Due to these limitations, there might be related work that is missing from this study.

3.5 Conclusion

Related works are model-based RE approaches with a scope on functional requirements that solve at least some of the four challenges that MIRA poses. From the challenges, we extracted a set of characteristics to identify and discuss the work related to MIRA. Related work

- presents a model-based RE approach.
- handles functional requirements.

[8] https://drive.google.com/file/d/0B77ALaznd8SBTmhvN2RQRUVtZ1U/view?usp=sharing, last accessed 11-29-2016

- facilitates the representation of both textual and formalized (heterogenous) requirements.
- facilitates a quality assurance of requirements.
- is classified as formal.

In a systematic mapping study, we screened more than 8000 papers on our inclusion and exclusion criteria to identify 336 papers on model-based RE. From these, 24 papers present related work. We identified a further 15 papers by snowballing and a search on new papers. Nonetheless, due to the threats and limitations of this study as described in Section 3.4 we could have missed related work. We discussed all related approaches in more detail.

Challenge 1: Heterogenous Requirements Specifications. Only 37 of the 336 papers in the SMS cover heterogenous requirements specifications. Two of the investigated approaches only facilitated the heterogenous representation of domain knowledge. The other papers covered the heterogenous representation of requirements similarly to MIRA.

Challenge 2: Analysis, Validation and Verification of Functional Requirements. Many of the examined related approaches focus on the analysis, validation or verification of requirements, whereas MIRA covers all three aspects of quality assurance.

Challenge 3: Seamless Model-based Development. The challenge that distinguishes MIRA most from its related work is the integration into a seamless model-based development approach. The related work either propagates an automated transformation from requirements to design or completely lacks an integration into a seamless development approach. MIRA proposes a manual transformation from requirements to design, thereby not forcing to predefine all design decisions.

Challenge 4: Guidance and Tool Support. Some papers present or discuss tool support for model-based RE. Nonetheless, as many approaches only focus on one or two aspects of quality assurance, guidance and tool support of these approaches is also limited to these aspects. The RE method support of many commercial modeling tools is typically limited to the RE capabilities of the supported modeling languages. Requirements management tools typically do not support requirements modeling. This requires practitioners and researchers to customize and connect requirements management and modeling tools to support model-based RE, which costs time and may lead to problems at the tool interfaces. MIRA contains a dedicated guidance and tool support for model-based RE.

The study could not identify an approach that covers all four challenges to the same extent as MIRA.

Chapter 4

Model-based Quality Assurance in RE

A requirements specification is not an end in itself, but the basis to develop a system that ideally meets the expectations of all of its stakeholders. A requirements specification documents these expectations so that it can serve as an input for the further development activities of the system under development. Through these development activities, the requirements specification can impact the quality of the system under development. We call those characteristics of a requirements specification that impact the system under development *system quality factors*. For example, completeness is a system quality factor because its absence (incompleteness) can lead to missing system behavior. Other characteristics of the requirements specification affect the development effort and associated risks, see also [vL01, p. 35].

Quality assurance (QA) aims at increasing the quality of an artifact. From an RE perspective, QA can investigate the quality of the requirements specification and it can ensure that the realization in the architecture and implementation conforms to the requirements. Model-based techniques promise to increase the effectivity and efficiency of QA, but require that the requirements specification has specific characteristics, for example, is represented formally. We call those characteristics of a requirements specification that have an impact on the effectivity and efficiency of QA activities *QA quality factors*.

Designing the requirements specification so that model-based techniques can be applied increases the effectivity and efficiency of QA. Effective and efficient QA may increase the quality of the system under development. A schematic overview of the impacts of QA quality factors on the system quality is given in Figure 4.1.

The aim of this thesis is to increase the quality of the system under development by supporting model-based QA. Therefore, in this chapter we perform a study to investigate the QA quality factors of a requirements specification that enable model-based techniques. The QA quality factors in scope comprise the contents of a requirements specification, their structure and representation form. This thesis has a scope on functional requirements, i.e., requirements on the system behavior. This leads to the following main research question:

> ***RQ1:*** *What are quality factors of a requirements specification with a positive impact on the effective and efficient model-based quality assurance of functional requirements?*

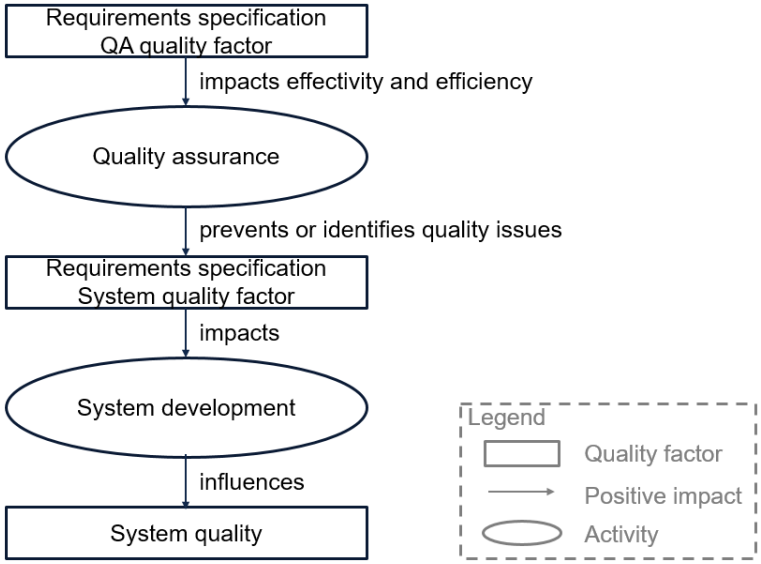

Figure 4.1: A schematic overview of the impacts between QA quality factors and system quality

The contribution of this chapter is a systematic investigation of QA quality factors of the requirements specification. The study investigates which information should be documented in a requirement specification to conduct QA in order to increase the *effectiveness* of QA. The *efficiency* of quality assurance should be increased by applying constructive QA to avoid quality issues already during the specification and by facilitating an automation of QA. For example, a formal representation does not only enable formal QA techniques that can be automated; a second, direct effect of the formalization are less ambiguous requirements. The study result are three matrices that capture the impacts of the QA quality factors on the QA activities.

The study results are the foundation to develop a model-based RE approach that enables quality assurance by supporting the impacts determined in this study. The model-based RE approach should provide rules and guidelines to ensure the QA quality factors in the requirements specification. A second essential aspect to be supported by the model-based RE approach is the automation of QA activities. Automation requires an adequate tool support. Therefore, Chapter 5 investigates the requirements of practitioners for a model-based RE tool.

Contents

4.1 Research Method

Study Design. The study instantiates an activity-based RE quality model. A *quality model* is "an abstraction of the relationships of attributes of the artefacts, process, and people and one or more quality attributes [here called quality factors] of the product" [Wag07]. An *activity-based RE quality model* [FMF15] investigates the quality of *artifacts* or *entities* contained in an artifact from the perspective of the *activities* performed on these artifacts. The entities have *quality factors* that can impact the activities positively (quality characteristics) or negatively (quality issues). The argumentation for the impact of a quality factor on an activity should be provided in a *rationale*.

In this study, the artifact under investigation is the requirements specification. An entity of a requirements specification could be for example a chapter or, more fine-grained, an RE concept. The more detailed the definition of an entity is, the more precisely we can discuss its impact on quality. An outline only facilitates to discuss whether a chapter might be relevant for QA. RE concepts facilitate a more detailed discussion about impacts. Therefore, this study investigates entities on the level of concepts. The activities under investigation are the QA activities discussed in Section 2.2, e.g., the validation of requirements. The impacts under investigation are effectivity and efficiency of a QA activity regarding the system quality factors, hence positive impacts. For example, an executable representation of a requirement has a positive impact on the validation of requirements, because it enables an effective simulation.

In the course of this study, the activity-based RE quality model is instantiated in a series of tables. An example of such a table is given in Figure 4.2. On the left hand side of the table, artifacts and concepts are listed. To its right, the QA quality factors of the artifacts and concepts are listed. In the upper part of the table, QA activities and subactivities are provided. In the middle part of the table, the impacts of QA quality factors on QA activities are documented. '+' or '++' capture a positive impact, the number in brackets provides the section in which the impact is discussed. '+' means that the impact is discussed in this thesis. '++' indicates an external source for the impact. Optionally, the impacted system quality factor can be documented. Figure 4.2 can be read as 'an executable requirement has a positive impact on the simulation, a subactivity of the validation, and can thereby increase the adequacy of the requirement (see Section 4.5.2).'

		QA Activities			
		Activity 1 e.g., validate		…	Activity n
		Subactivity 1.1 e.g., perform simulation	Subactivity 1.2		Subactivity n.1
		System quality factors e.g., adequacy			
Artifacts & RE concepts	**QA Quality Factors**	**Impacts**			
Requirements specification	QA quality factor A				+ (4.5.1)
Concept of the requirements specification e.g., requirement	QA quality factor C e.g., executable	++ (4.5.2)			
	QA quality factor B		+ (4.5.2)		

Figure 4.2: Example of a table that instantiates the activity-based RE quality model

Research Questions. The research question addressed by this study help to elaborate the quality model, see Figure 4.3:

> ***RQ1.1:*** *What are system quality factors of the requirements specification?*

In the first step, we investigated the system quality factors that the QA activities should improve. We limited the study to those factors that impact the system under development, leaving out factors that only effect development efforts.

> ***RQ1.2:*** *Which concepts of a requirements specification enable which quality assurance activity?*

In the second step, we searched for concepts to be contained in the requirements specification that enable QA activities or subactivities. The result of this investigation is a table that captures the impacts of these concept on QA activities.

> ***RQ1.3:*** *Which QA quality factors of a requirements specification facilitate which automated analytical quality assurance activity with which impact on a system quality factor?*

In the next step, we investigated model-based techniques that can be used to automate the QA activities in order to increase their efficiency. We determined the QA quality factors that are necessary for conducting the techniques and the activities that can be automated. Furthermore, we investigated the effectiveness with respect to the system quality factors. A table documents the impacts of QA quality factors on QA activities through automated model-based techniques.

> ***RQ1.4:*** *What are the constructive impacts of the QA quality factors that facilitate automated quality assurance on the system quality factors?*

Creating a requirements specification that has certain QA quality factors may not only enable an automation of QA, but may also have a constructively improve the system quality factors. We investigate this impact for those QA quality factors that facilitate automated quality assurance. The result is a table that captures the impacts of the QA quality factors discussed in research question RQ 1.3 on constructive QA.

Figure 4.3 shows which parts of the quality model are filled by which research question. The set of system quality factors in scope is determined in RQ1.1. In RQ1.2 to RQ1.4 we investigate these system quality factors or subsets thereof. For each of the research questions RQ1.2 to RQ1.4 we develop a partial quality model. We then consolidate the partial models into one quality model that presents the study result.

Data Collection. The study is based on a set of well-established system quality factors defined by van Lamsweerde [vL09] with a clear impact on the quality of the system under development.

The main QA activities were introduced in Chapter 2. Van Lamsweerde [vL09] presents a set of model-based QA techniques that can automated and that support

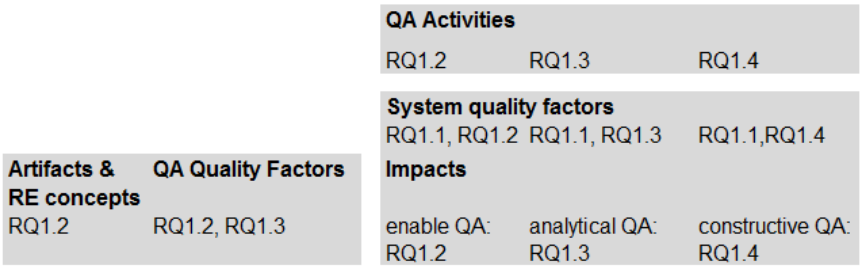

Figure 4.3: The research questions for elaborating the quality model

QA activities together. He also presents necessary QA quality factors to conduct these techniques. These QA activities, techniques and quality factors are the input for this study. The main RE concepts have been investigated based on the concepts defined in AutoRAID [Sch09] and in the SPES requirements viewpoint [DTW12]. The impacts of QA quality factors on QA activities and thereby on the system quality factors have been elaborated based on a literature review and amended with own work. Evidences for impacts of the quality factors on activities are extracted from literature and amended with own work. For traceability and to facilitate refutability, the sources of evidences are provided in the argumentations. It is labeled when evidences stem from case studies or experiments. Benefits of similar approaches have been aggregated, for example, benefits of similar modeling languages. Nonetheless, the differences are made clear in the description of the results. An overview of the data sources is provided in Figure 4.4.

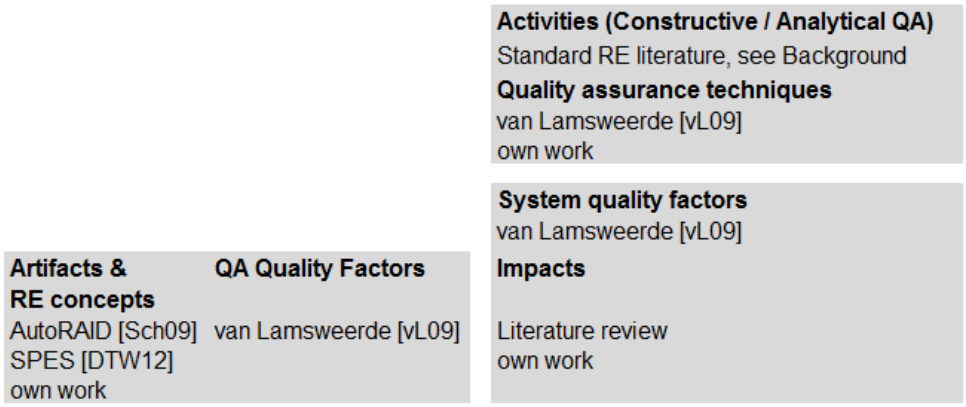

Figure 4.4: The sources for elaborating the quality model

The resulting tables summarize quality factors, activities and impacts, where supporting evidence was identified in the course of this study. There may exist further impacts between QA quality factors and activities, even if they are not discussed in this study.

The study investigates a limited set of quality factors, activities and impacts and faces some threats, see Section 4.2. Section 4.3 discusses the set of system quality factors in scope of this study (research question RQ 1.1). The subsequent chapters instantiate the activity-based RE quality model with respect to the research questions. Section 4.4 investigates the concepts that should be documented in a requirements

specification for quality assurance (research question RQ 1.2). Based on these concepts, Section 4.5 investigates quality factors with an impact on analytical quality assurance (research question RQ 1.3). Section 4.6 investigates quality factors with an impact on constructive quality assurance (research question RQ 1.4). Related work is provided in Section 4.7.

4.2 Threats to Validity and Limitations

The study faces the following threats to internal and external validity and limitations.

4.2.1 Internal Validity

Heterogeneous Definition of Quality Factors and Activities. A threat to validity is the heterogeneous definition of quality factors and activities. For example, the terms 'validation' and 'verification' may have an inverted definition in some publications compared to the definition in this work. The threat is addressed by refining quality factors and activities in this study and by mapping the terms to the definitions provided in this work.

Reliability of Impacts. A threat to validity concerns the impacts of quality factors on activities. These impacts may be wrong. This study could identify case studies and experiments that investigated impacts empirically, but not for all impacts. This threat of the reliability of impacts is addressed by providing the argumentation and, whenever available, the sources in literature on impacts. To decrease researcher bias and to increase the reliability of the impacts, all argumentations have been discussed with several experienced practitioners and researchers. Furthermore, an industrial case study in the train automation domain in Chapter 9 confirms some of the presented impacts using the model-based RE approach developed in this thesis to provide further evidence.

4.2.2 External Validity

Generalizability. The generalizability of the impacts differs. Some impacts are trivially true whenever a quality factor is mandatory for a QA technique. Some impacts are based on an argumentation that provides a rationale why a quality factor has an impact on an activity; this argumentation may only hold in a specific environment. Some impacts have been investigated in studies for a specific specification modeling language, environment or system. The generalizability of the findings of these studies to other modeling languages, environments, or systems needs further investigation. Sources for all studies and the rationales have been presented. As mentioned above, a case study presented in Chapter 9 confirms some of the presented impacts for a specific modeling language for a train automation system. Nevertheless, the generalizability remains an open research question.

4.2.3 Limitations

Limited Set of QA Activities. This study investigates constructive and analytical quality insurance. For constructive QA, the study investigated the requirements specification, comprising the documentation, formalization and tracing, see Chapter 2. Constructive QA for the elicitation was excluded. Analytical quality assurance comprises the analysis, validation and verification.

Further supporting activities for quality assurance have been excluded consciously: For example, human-centered activities concerned with collaboration and communication might influence the success of quality assurance. A pragmatic reason to not apply formal techniques is the lack of time of skilled requirements engineers. Stakeholders that are not trained on a formal modeling language may find it difficult to express, read and understand these requirements; they need a natural language representation.

Furthermore, the quality factors discussed in this study can impact activities aside quality assurance. The quality factors discussed in this study may also impact the maintainability of the requirements specification. A well-structured specification may have a positive impact on the change impact analysis.

As future work, the impact of the quality factors on these activities may be analyzed and included into the study results.

Limited Set of Analytical Quality Assurance Techniques. The set of analytical QA techniques is limited to those techniques listed in [vL09, p. 187ff]. These techniques require structuring the requirements specification according to rules that can be captured in models. Other approaches have been developed for the automatic QA in unstructured text, for example, to detect redundant requirements [FCC13]. Exploring QA quality factors that impact this kind of automatic QA is outside the scope of this work.

Limited Set of Concepts. The concepts discussed in this work are limited to a subset of the concepts presented in ARAMiS [PE12] and SPES [DTW12]. Further concepts may be required to conduct specific quality assurance techniques other than the ones presented by van Lamsweerde [vL09] or to address further system quality factors.

Limited Set of Quality Factors. The QA quality factors are limited to those factors that enable the QA techniques presented by van Lamsweerde [vL09, p. 187ff]. The system quality factors are limited to those factors presented by van Lamsweerde [vL09, p. 36ff] that affect the quality of the system under development. This list excludes system quality factors that only may influence development effort and risk.

Limited Set of Impacts. This study investigates a limited set of impacts. If no impact is provided between a QA quality factor and an QA activity that does not imply that no impact exists. It only means that here no impact has been identified in the course of this study.

Limitation to Project-Independent Quality. The relevance of the quality of the requirements specification depends on particular project characteristics [MMFFE15]. This means that the project context influences costs and risks of quality issues compared to the benefits of a high quality. However, the RE research community still lacks a deep understanding of the usage of requirements specifications in practice and the degree to which the quality of a requirements specification impacts subsequent development activities [MMFFE15]. Therefore, this work investigates the quality factors of a requirements specification independent of a specific project context. As this work does not focus on a specific project context, the investigation of costs, risks and benefits of a high quality in the requirements specification are outside the scope of this work.

4.3 System Quality Factors

In answer to research question RQ1.1., this section classifies a set of factors from two sources into quality factors, impacts and activities as defined in the activity-based RE quality model introduced in Section 4.1. We discuss and detail the factors presented by van Lamsweerde [vL09, p. 36ff] and the widely used [SB13] standard ISO/IEC/IEEE 29148:2011 [ISO11b]. We will see that some of the factors constitute system quality factors, other factors only define activities that need to be conducted during the system development or impacts on these activities that need to be achieved.

4.3.1 Adequate

"The requirements must address the *actual* needs for a new system – explicitly expressed by stakeholders or left implicit. The software requirements must be adequate translations of the system requirements [...]. The domain properties must correctly describe laws in the problem world. The environmental assumptions must be realistic" [vL09, p. 35]. Other authors call this quality factor *correct* [ZG03]. An inadequacy may lead to an *implementation* that meets its requirements, assumptions and domain properties, but that are not the right ones [vL09, p. 37].

4.3.2 Unambiguous

"The requirements, assumptions and domain properties must be formulated in a way that precludes different interpretations. Every term must be defined and used consistently" [vL09, p. 35]. Ambiguity may prevent stakeholders to understand the meaning that was intended by the author. Following van Lamsweerde [vL09, p. 37], ambiguous entities of the requirements specification allow an interpretation of statements contained in the requirements specification in different ways; ambiguous statements in the requirements specification may result in an *implementation* of a system built on the interpretation of requirements, assumptions or domain properties that are different from the intended interpretation.

In the field of linguistics, a distinction is made between different types of ambiguity [Cona]:

- *Lexical ambiguity* means that a word has multiple meanings. An example is the term *bank* which can denote a financial institution or a river side.
- *Syntactic ambiguity* of a sentence caused by alternative interpretations of the structure of that sentence. An example is the phrase *We saw the man with the telescope*.
- *Semantic ambiguity* occurs when a sentence contains a word or a phrase with multiple possible interpretations in the context of that sentence. The sentence *We saw her duck* allows to interpret *her duck* either as a bird or as an activity [HHS07].

This distinction facilitates to further specify the 'different interpretations' of a reader of a requirements specification.

4.3.3 Complete

"The requirements, assumptions and domain properties, when taken together, must be sufficient to ensure that the system-to-be [the system under development] will satisfy all its objectives. These objectives must themselves be fully identified, including quality-related ones. In other words, the needs addressed by the new system must be fully covered, without any undesirable outcomes. In particular, we must have anticipated incidental or malicious behavior of environmental components so that undesirable software effects are ruled out through dedicated requirements. A requirement on software behavior must prescribe a desired output for all possible inputs. The specification of requirements and assumptions must also be sufficiently detailed to enable subsequent software development" [vL09, p. 35]. Omitting statements in the requirements specification may result in an *implementation* that does not take into account these statements [vL09, p. 36].

The definition of van Lamsweerde indicates the notion of external completeness. Zowghi and Gervashi define *external completeness* as "external completeness ensures that all of the information required for problem definition is found within the specification" [ZG03]. This notion of completeness "clearly demonstrates why it is impossible to define and measure absolute completeness of specifications. The only truly complete specification of something would be the thing itself" [ZG03].

Behavioral completeness describes the completeness of a system with respect to its input-output behavior. Behavioral completeness can be defined by a formal description of input-output relations that prescribe expected behavior. Van Lamsweerde [vL09, p. 203] suggests to require such relations to be functions where for every input situation should exist at most one corresponding output to avoid non-deterministic behavior. Requiring such functions to be *total* [vL09, p. 203] ensures that an output for every input situation exists (surjective function). Often, input-output relations are defined on the input history [HL96, BS01].

The activity-based view on the requirements specification adds a third notion of completeness: Activity completeness assesses completeness from the perspective of software and systems engineering activities. A requirement or requirements specifica-

tion is *activity complete* with respect to a specific development activity if it comprises sufficient information to perform that activity. For example, 'performance completion' of a specification implies that it contains enough information to evaluate the performance of a system [WPS02]. The required information is highly dependent on the concrete development activity. Development activities comprise activities performed in RE as well as activities from other development phases such as design, implementation and test. Activity incompleteness may lead to the inability to perform an activity or may influence its effectivity. Depending on the activity, both may impact the system under development. Other activities may only impact development effort and risks. It depends on the concrete activity, whether activity completeness represents a system quality factor. This chapter discusses the activity completeness for quality assurance activities. A reduced effectivity of quality assurance can affect the quality of the system under development. Therefore, activity completeness for quality assurance is a system quality factor.

4.3.4 Consistent

Requirements are elicited from different sources and viewpoints. These diverse sources and viewpoints can include contradicting expectations on the system under development. "The requirements, assumptions and domain properties must be satisfiable when taken together. In other words, they must be compatible with each other" [vL09, p. 35]. If a set of requirements, assumptions or domain properties contradicts each other, then it is impossible to produce a correct *implementation* [vL09, p. 36].

Logical conditions for *logical consistency* and inconsistency can be precisely defined when expressing requirements, assumptions and domain properties as logical assertions [GKvdBV11], [Bro13a], see also Chapter 2. These conditions reveal statements that may not be implementable. Nonetheless, if the source of a conflict, for example a domain property, is not documented (not externally complete), or is an unrealistic assumption, then logical conditions may not identify inconsistencies.

Feldmann et al. [FHK+15] describe *type inconsistencies* as arising from using incompatible values and types. They also include improper type conversions and impossible type conversions. A prominent example for the negative impact of type inconsistencies is the failure of the Mars Climate Orbiter; conversion errors from English imperial units to metric units resulted in a loss of the orbiter[1].

4.3.5 Investigation of Further Quality Factors

Besides the system quality factors discussed above, van Lamsweerde introduces further factors. These factors do not directly impact the system under development and therefore do not constitute system quality factors. *Pertinence* requires that all requirements contribute to a goal. The absence of pertinence could lead to increased effort when finding out that a requirement is not needed. All other quality factors can be directly related to activities: *Measurability* means that the requirements are stated in a way that enables their testing/verification; *feasibility* facilitates the realization

[1] https://mars.jpl.nasa.gov/msp98/orbiter/

in budget, schedule and technical constraints; *comprehensibility* directly refers to the comprehension by the users of the requirements when reading and processing the requirements; *good structuring* defines the organization of the requirements specification; *modifiability* defines a set of modification activities; *traceability* directly refers to the tracing activity.

The ISO 29148 also provides definitions for all system quality factors, for an unambiguous, complete and consistent requirement as well as for complete and consistent sets of requirements. Similar to the notion of an adequate requirement, the standard defines the quality factor *necessary*. Further factors do not directly impact the quality of the system under development, but rather impact the development efforts and risks. The quality factor *bounded* relates requirements to the scope of the system, similar to the notion of a pertinent requirement. Related to the implementation of the system under development, the standard defines *implementation free* and *feasible* requirements. Related to the costs of implementation is the quality factor *affordable*. Other quality factors are directly related to activities, namely *verifiable* and *traceable* requirements. The definition of a *singular* requirement does not provide any hints on its benefits for any activity.

Without the notion of an activity-based RE quality model, the effects of many of these quality factors on the system under development remain unclear. Most quality factors discussed here define and describe an activity on the requirements specification (e.g., *traceable*) that should be performable. In an activity-based quality model, the quality factors that define activities would rather be defined as activities and could be refined to sub-activities and tasks. *Pertinence* is a quality factor in the sense of the activity-based quality model with a clear impact (increased effort) on the implementation. In an activity-based quality model, the quality factor *affordable* would be a type of impact, not a quality factor. To map the quality factor *singular* to the activity-based quality model requires further information.

4.3.6 Summary

The investigation answered research question RQ1.1 and identified a set of four *system quality factors*. Van Lamsweerde [vL09, p. 36f] provides an argumentation about the effects of these factors. We further detailed these system quality factors:

- Adequacy
- Lexical, syntactic, and semantic unambiguity
- External, behavioral, and activity completeness
- Logical and type consistency

This list is relevant, as many scientific publications in the last decades discuss these system quality factors [JLHM91, HJL96, ZG02, NER00, SBHW03, Fir05, dSAVP10]. Furthermore, an extensive interview study with experts from 30 German companies [FW13] revealed that incomplete / hidden requirements and inconsistent requirements are two problems in the (German) industry that have been acknowledged by most of the participating practitioners (indicated by a mean and mode value of 4 on an ordinal scale, where 5 is “I agree” and 1 “I disagree”). Nonetheless, the list of quality factors is non-exhaustive; other quality factors may exist that have an equally

severe impact on the system under development.

An interesting finding of the investigation is that some of the factors mentioned by van Lamsweerde or in the ISO 29148 would not be classified as a quality factor in an activity-based quality model. They would rather be classified as activities or impacts. As future work, a broader systematic investigation could be conducted using the activity-based quality model. This investigation could a) classify factors from literature to the activity-based quality model on a bigger scale, and b) identify more quality factors of the requirements specification that impact the quality of the system under development.

4.4 Concepts of the Requirements Specification

In order to answer research question RQ 1.2, a set of RE concepts are introduced. Each concept has an immediate impact on specific QA activities when it is instantiated. Explicitly *pre-defining* these RE concepts supports the person that creates the requirements specification or parts thereof to document these. Thereby, the person that performs QA can benefit from these impacts. Furthermore, defining RE concepts has immediate merits on constructive and automated QA. With corresponding tool support, the contents of a requirements specification can be stored according to these definitions. This facilitates to automatically access and evaluate the information. The precise definition of RE concepts facilitates to define further essential characteristics that are necessary to conduct automated QA, for example, representation forms. These benefits are explicated in the subsequent chapters. In the following, we present the RE concepts and describe their impact on the QA activities.

4.4.1 Goals

A goal is “an objective the system under consideration should achieve" [vL01]. Thereby, goals are stakeholder intentions that “represent a first manifestation of the stakeholders’ system vision” [DTW12, p. 55]. For example, the goal of a pedestrian using a traffic light system is to safely cross the street. Documenting goals can support the quality assurance: Goals provide a criterion for the external completeness of system requirements [vL01]. Assuming that the set of goals is complete, then the requirements are complete, if all goals can be achieved from the requirements and the considered domain properties. Unrefined goals may indicate missing requirements. “Goals give rationales and justifications for the functionality and features a system must possess” [DTW12, p. 55]. Access to these rationales and justifications enables the stakeholders to understand the rationale of a system requirement. Thereby, goals support the validation of requirements.

4.4.2 System Requirements

A system requirement is a prescriptive statement to be enforced by the system under development. System requirements form the basis for the design, implementation and quality assurance of the system under development. Defining and documenting

system requirements is a precondition for their (semi-)automated analysis, validation and verification that is discussed in Section 4.5. It is commonly distinguished between requirements concerning the functionality of a system and others [Gli07]. A study [EVMF16] investigated why practitioners distinguish between functional and extra-functional requirements. According to the study, the main differences in QA processes are: 1.) Different stakeholders are involved in development activities on the requirements; 2.) the quality assurance techniques differ; 3.) extra-functional requirements need to be monitored continuously during the implementation in contrary to functional requirements. Practitioners distinguish them because they want to achieve more complete and less ambiguous requirements. The study could not confirm a clear positive impact of a distinction on quality assurance. Practitioners that distinguish functional and extra-functional requirements, claim that this leads to missing testability; others state that this results in focused tests and explicit test for extra-functional requirements. Practitioners that do not distinguish them, claim that validation and verification suffer; other state that the tests are more comprehensive. Summarizing, from a project-independent viewpoint, the impact of distinguishing functional and extra-functional requirements on quality assurance needs to be investigated further for specific QA techniques.

4.4.3 Use Cases and Scenarios

(System) use cases describe an excerpt of interactions of a system under development with its actors for a certain case of use. The system thereby responds to the request of one of the actors in order to achieve a particular *goal*. Use cases can be detailed by *scenarios*. Scenarios describe the steps from the trigger to goal delivery, inclusive any "clean-up" [Coc00] afterwards. They can be described depending on the particular requests and conditions surrounding the requests. A use case collects together those different scenarios [Coc00]. For example, a use case may collect all scenarios that describe how a pedestrian uses (or missuses) the traffic light system to cross the street. Use cases and scenarios help stakeholders to validate the goals that they cover [vL01].

4.4.4 Refinement Links

A refinement link documents the refinement from and within goals, use cases and system requirements. As discussed above, the refinement from goals to use cases, scenarios and system requirements enables specific quality assurance activities. When these refinements are documented explicitly as refinement links, everybody who conducts quality assurance on the requirements specification can access the information about refinements at all times.

4.4.5 External Trace Links

External trace links connect requirements with other development artifacts, for example, a concept of the system architecture. Thereby, external trace links enable verification: When requirements are derived from other artifacts, this link can be evalu-

ated to verify the requirements against this artifact. When an artifact is derived from requirements, the link can be evaluated to verify the artifact against its requirements.

4.4.6 Glossary

A glossary provides definitions of the domain-specific terms that are used in the requirements specification. Thereby, a glossary supports persons involved in the quality assurance activities to have a common understanding of the concepts behind these terms. *Lexical ambiguity* can be caused by a word with different meanings (homonyms) and by different words with the same meaning (synonyms) [Poh10]. Defining the meaning of terms used in the requirements specification in a glossary can decrease these kinds of ambiguity [Poh10].

4.4.7 Requirement Sources

Each requirement has a source from where it origins. These sources can be documented as requirement sources. The requirement sources can be differentiated in three types: Stakeholders, documents and external systems [Poh10]. A *stakeholder* is anyone with an interest in or an effect on the outcome of the system under development [RR06]. An instantiation of a stakeholder of the traffic light system is the pedestrian that uses this system to cross a street. To elaborate all instantiations of a stakeholder, all persons with an interest in or an effect on the outcome of the system under development have to be identified. *Documents* include for example standards or specifications of similar or preceding systems. Further requirements can stem from those *external systems* that the system under development should interact with. Pohl [Poh10] discusses the positive impact of requirements sources on a more externally complete requirements specification. Documenting requirement sources can also be valuable during the manual quality assurance of requirements as they may be the rationale for a requirement. Analogously to glossary terms, when stakeholders have access to a definition for each requirement source, then this can decrease *lexical ambiguity* [Poh10] and ensure a common understanding of the relevant requirement sources amongst the stakeholders.

4.4.8 Cross-References

Glossary terms and requirement sources provide term definitions on the domain-specific context of a system under development. These term definitions can help to improve the understanding of those persons that are reading the requirements. Stakeholder read requirements as part of a manual quality assurance. A cross-reference connects the domain-specific terms used in a requirement with their term definition as glossary terms and requirement source. For example, goals state the intent of one or more stakeholders and may document this stakeholder. Therefore, a goal could have a cross-reference to this stakeholders. An alternative would be to have redundant term definitions in the specification. Less redundant information may lead to less inconsistencies, because each piece of information has to be documented and changed only once. A study [JDF+10] investigated the impact of redun-

dant information in form of clones on development activities. The study was performed on 28 software requirement specifications from various domains including embedded systems. The authors of the study concluded that cloning significantly increases the effort for activities that involve reading the requirements specification, for instance, during inspections. Changing duplicated information is costly and error-prone. Cloned fragments of the requirements specification can lead to clones in the code or developing the same functionality repeatedly; cloned code may lead to *inconsistent changes* that may even lead to system failures. The authors concluded that to prevent the negative consequences of clones, redundancy in a requirements specification should be avoided.

4.4.9 QA Results

After the application of a quality assurance technique, the *QA results* have to be documented if they are not resolved immediately. For example, QA checklists that capture the applied activity and its results could be defined in addition to the requirements specification. These checklists should be traced to the investigated contents. Specific trace link types can be used to document quality issues affecting several requirements for further processing. An example are inconsistency trace links to document inconsistent requirements.

4.4.10 Summary

In answer to research question RQ 1.2, we investigated a set of RE concepts such as goals and use cases that enable specific quality assurance activities. The activities are concrete quality assurance checks, for example, check the completeness of requirements, or are subactivities, for example, to understand the rationale of a requirement. Pre-defining the RE concepts facilitates the requirements engineer (or any person that writes a specification) to instantiate them and make use of the positive impact on the quality assurance activities. The impacts of these concepts on quality assurance activities are summarized in Figure 4.5.

4.5 Automated Analytical Quality Assurance

In order to answer research question RQ 1.3, we analyze the model-based quality assurance techniques presented by van Lamsweerde [vL09]. We investigate the factors that the requirements specification must exhibit in order to enable an automation of these techniques.

Analytical quality assurance consists of several subactivities, where one or all of these subactivities can be automated: Firstly, the objects are selected from the requirements specification that should be investigated. Secondly, we need to apply the concrete quality assurance technique on these objects. Simulation, database queries and formal techniques are some of the main analytical techniques for quality assurance in RE that can be automated [vL09, p. 187ff]. According to a survey amongst 419 software projects [FGZ15], the majority of these projects used manual inspections and

		Analytical QA activities					
		Analyze	Validate		Verify	Quality assurance	
		Check completeness of requirements [vL01]	Understand rationales for requirements [DTW12]	Validate goals [vL01]	Identify verification object	Gain common understanding of terms	Document QA results
		System quality factors					
Artifacts & RE concepts	**QA Quality Factors**	+ external completeness **Impact**				+ lexical unambiguity	
Goal	Pre-defined concept	++ (4.4.1)	++ (4.4.1)	++ (4.4.3)			
System requirement	Pre-defined concept	++ (4.4.1)	++ (4.4.1)		+ (4.4.5)		
Use case / scenario	Pre-defined concept			++ (4.4.3)			
Refinement link	Pre-defined concept	+ (4.4.4)	+ (4.4.4)	+ (4.4.4)			
External trace link	Pre-defined concept				+ (4.4.5)		
Glossary	Pre-defined concept					++ (4.4.6)	
Requirement source	Pre-defined concept	++ (4.4.7)	+ (4.4.7)			++ (4.4.7)	
Cross-reference	Pre-defined concept					+ (4.4.8)	
QA result	Pre-defined concept						+ (4.4.9)

Figure 4.5: Summary of the impacts (++, +) of pre-defining RE concepts on QA activities; in brackets behind the impacts are the sections where the impacts are discussed

reviews, 8% of the projects applied simulation and 7% automated checking on a formal specification of the system. This survey did not include database queries into the list of answers to select. Finally, the results are documented; this is necessary, if the technique is automated or the resolution of the detected issues should be decoupled from the resolution. The resolution of detected quality issues is out of scope of this work. The quality assurance techniques and the subactivities to perform them are depicted in Figure 4.6.

Automated techniques require a specific structure and representation of the information contained in the requirements specification. For example, a requirement has to be represented using a formal modeling language to apply formal techniques. This structure and representation directly positively impacts the quality assurance activities as it enables automated analysis, validation and verification techniques. Therefore, the structure and representation of a requirements specification are the QA quality factors under investigation.

4.5.1 Automated Queries on a Requirements Database

Automated queries on a requirements database require that the requirements specification is *structured by an underlying data model* and that the requirements specification is stored in a database according to that data model. A machine-readable definition of

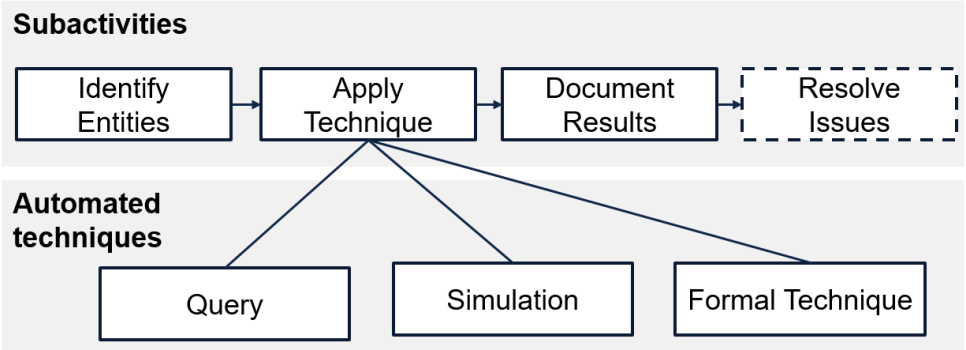

Figure 4.6: Applying automated quality assurance techniques

the requirements specification facilitates queries (with corresponding tool support). Queries on the database can then automate inspection tasks that otherwise would have to be conducted manually. This machine-readable definition of the structure of the requirements specification can be achieved by several means. A means to enable queries is to pre-define the entities contained in the requirements specification. Furthermore, representing the entities using a semi-formal or formal modeling language enables syntactic or semantic checks via database queries.

Select Contents. A subactivity common to all QA activities is to select the objects contained in the requirements specification that should be investigated. Queries can automate the task of extracting and preparing the data needed for further inspection. The *definition of RE concepts* enables to store the contents of a requirements specification according to these definitions. If each object is annotated with its type, a query can refer to these entities. A query then can select objects based on the definitions. Section 4.4 discussed a set of concepts to support QA activities that are candidate inputs for such a query. *Trace links* capture the dependencies within the entities contained in a requirements specification and from the requirements specification to other development artifacts. Trace links include cross-references from stakeholders to their requirements, refinement links between requirements, and external trace links between requirements and the system architecture specification. These trace links can be evaluated in queries. For instance, the definition of goals, system requirements and refinement links between them supports the completeness check on system requirements. To support this analysis, a query can extract all system requirements that refine a goal by evaluating the refinement links. The result of this query is the input to check the system requirements for completeness. A second example are stakeholders that want to validate their requirements using simulation. Stakeholders and requirements are documented including cross-references from stakeholders to their requirements. Then the stakeholders could identify the relevant subset of requirements using these links. If the information is stored so that it is machine-readable, then the relevant set of requirements can be extracted by a query. The definition of RE concepts facilitates to *define attributes* on these concepts to increase the preciseness of queries. For example, a quality assurance status enables to determine only those entities that need further analysis.

Analyze Consistency and Completeness. Queries can be used to automatically perform *conformance constraints* that *analyze* the structural conformance of the requirements specification to a set of rules. A language to define conformance constraints is OCL [OMG06]. Schätz [Sch09] defines conformance constraints for automating completeness and consistency analyses. Schätz defines consistency conditions on the RE concepts to be contained in the requirements specification, their attributes and trace links. Similarly, conformance constraints can be defined on the RE concepts, including the trace links discussed in Section 4.4. In addition, van Lamsweerde [vL09, p. 241f] discusses queries on semi-formal and formal representations of parts of the requirements specification.

4.5.2 Semi-automated Simulation

If parts of the requirements specification are *executable* or can be transformed into an executable form, then this part can be simulated [vL09, p. 187ff]. Simulation is a common means for the *animation-based validation* of the requirements specification. With the help of an animation tool, various events to which the system could be exposed can be re-enacted in order to *validate* whether the response of the model is adequate. For instance, Gaucher and Jeannet [GJ16] report on the successful application of simulation for the validation of requirements in the STIMULUS tool, detecting ambiguities and omissions.

4.5.3 Formal Techniques

Mathematical techniques that can be performed on a *formal specification* are called formal techniques. To use advanced mathematical quality assurance techniques, requirements need to be expressed in a formal modeling language that uses a precise, mathematical notation and forms part of a formal system modeling theory. The system modeling theory enables a formal reasoning that can be automated with corresponding tool support. For automated formal analysis and verification, the modeling language needs to be translated into the input language of a model-checker, for example NuSMV[2], or a theorem prover such as Isabelle[3].

Analyze Consistency and Completeness and Verify Adequacy. Formal techniques enable checks such as type consistency checks [vL09, p. 202f], logical consistency and behavioral completeness checks [vL09, p. 203] and deductive verification that a behavioral model satisfies a desired property [vL09, p. 205ff]. A formal verification of requirements and subsequent development artifacts prerequisites a common comprehensive system modeling theory that spans both the formal representation of requirements and subsequent development artifacts. This system modeling theory is "the theoretical basis to ensure a thorough formalization of all artifacts produced during the development of a system" [BFH+10]. A modeling paradigm that enforces this comprehensive modeling theory is *seamless model-based development* [BFH+10].

[2] Available at http://nusmv.irst.itc.it/, last accessed 9.5.2016
[3] Available at https://isabelle.in.tum.de/, last accessed 9.5.2016

Formal techniques have been used for example by Rockwell Collins for the formal analysis and verification of requirements for a flight guidance system [MTWH06] based on a formalization to the Architecture Analysis and Design Language (AADL) [FG12]. They identified a logical inconsistency between requirements and a mismatch between requirements and the implementation. As a consequence of discussions with domain experts, the requirements had to be changed in order to correspond to the implementation.

A comprehensive system modeling theory does not require to formalize requirements and design at the same level of detail. A *formal refinement specification* [BS01] is a means to define the mapping of the syntactic interfaces of two formal models. A formal refinement specification is a means to connect formal requirements on different levels of detail or to connect formal requirements and a formal model of the architecture. For example, at the requirements level, an input of the system would be defined as 'the user activates the traffic light system', formalized in the Boolean variable $tlc\_activated$. At the architectural level, the according system input stems from a sensor, a button ($button1$), and is modelled as the integer input variable $sensor\_value\_button1$. The formal refinement specification would define the mapping between $tlc\_activated$ and $sensor\_value\_button1$. By providing the mapping from requirements to design, a formal refinement specification facilitates a design-independent formal representation of requirements. A design-independent formal representation facilitates an early formal analysis of requirements before any architectural model is developed. Design-independent formal requirements do not need to be adjusted, when the architecture changes. Nevertheless, the formal refinement specification needs to be adjusted in this case.

4.5.4 Document Results

Automated quality techniques often only detect quality issues, but do not resolve them automatically. Therefore, the identification and resolution are decoupled and quality assurance results need to be documented automatically.

4.5.5 Summary

This section discussed three model-based techniques, database queries, simulation and formal techniques. For each technique, the necessary QA quality factors have been discussed that the requirements specification or some of its concepts must exhibit. In answer to research question RQ 1.3, the table provided in Figure 4.7 summarizes the study results for automated QA. The table summarizes the concepts and their QA quality factors that enable automated model-based QA. The impacts show, which QA factors enable which activities and which system quality factors can be improved.

		Analytical QA Activities					
		Select relevant contents Perform query	Analyze Perform query [vL09]	Apply formal technique [vL09]	Validate Perform simulation [vL09]	Verify Apply formal technique [vL09]	Document results to decouple identification and resolution
		System quality factors	+ logical consistency and behavioral completeness [vL09, Sch09]	+ type consistency [vL09] + logical consistency and behavioral completeness [vL09] + logical consistency and adequacy [MTWH06]	+ adequacy + unambiguities and completeness [GJ16]	+ adequacy [MTWH06] + adequacy of a behavioral model to a desired property [vL09]	
Artifacts & RE concepts	**QA Quality Factors**	**Impacts**					
Requirements specification	Pre-defined concepts	+ (4.5.1)	++ (4.5.1)				
RE concept	Pre-defined attributes	+ (4.5.1)	++ (4.5.1)				
System requirement	Semi-formal representation		++ (4.5.1)				
	Executable representation				++ (4.5.2)		
	Formal representation		++ (4.5.1)	++ (4.5.3)		++ (4.5.3)	
Trace link	Defined concept	+ (4.5.1)	++ (4.5.1)				
	Formal refinement specification					++ (4.5.3)	
QA results	Pre-defined concept						+ (4.5.4)

Figure 4.7: Summary of the positive impacts (++, +) of QA quality factors on automated QA techniques; in brackets behind the impacts are the sections where the impacts are discussed

4.6 Constructive Quality Assurance

The QA quality factors discussed in Section 4.5 impact the effectivity and efficiency of QA by automation. Creating a requirements specification with defined concepts, attributes and representation forms may already have a positive effect on the system quality factors. Hence, this section investigates these QA quality factors to determine those positive impacts in answer to research question RQ 1.4.

4.6.1 Pre-defined Entities

Pre-defining a set of mandatory *RE concepts* can increase *activity completeness* of the requirements specification with respect to development activities. Each of these RE concepts to be contained in a requirements specification should be related to a development activity. A thorough analysis of all RE concepts with respect to their use in the development process reduces unnecessary documentation effort. A thorough analysis of the development process with respect to undefined RE concepts reduces the risk of undocumented or unconsolidated information. Furthermore, for each RE concepts, the QA quality factors can be determined that are necessary for a specific QA technique. For example, formal verification of requirements requires a formal representation. The precise *definition of RE concepts* that should be instantiated as concrete objects in the requirements specification facilitates a guided *documentation*; the contents of the requirements specification are elaborated according to these definitions. For example, VOLERE defines the main entities to be contained in a requirements specification; the authors recommend the use of this template as a checklist in manual reviews to determine missing requirements [RR06, p. 323ff].

4.6.2 Pre-defined Attributes

"To support requirements analysis, well-formed requirements should have descriptive attributes defined to help in understanding and managing the requirements" [ISO11b]. Attributes guide and restrict the instantiation of each concept of the requirements specification, ensuring that it has the necessary constituents for the development activities performed on an entity. A structured description of a concept by pre-defined attributes can further improve *activity completeness* of development task by defining the concrete information to be documented for a concept. Tiwari [TG14] gives indications by a comparative study that a template that defines a more detailed set of attributes for use cases yields more complete and less redundant use cases in comparison to a less detailed template. An example for a positive impact on QA efficiency is an attribute that defines the QA status of a requirement. This status indicates whether a requirement has been fully specified, analyzed, validated or verified.

Enumerations and other more rigorous *data types* can further increase the quality of a requirements specification constructively by avoiding *type inconsistencies*. A uniform definition of data inputs is particularly valuable when data is entered by multiple users.

4.6.3 Pre-defined Trace Links

The fundamental concepts for the entities concerned in the *tracing* activity are based on Gotel el al. [GCHH+12b]. A *trace artifact* is a "traceable unit of data (e.g., a single requirement, a cluster of requirements, a UML class, a UML class operation, a Java class or even a person)" [GCHH+12b]. A trace artifact may be further described by its type. A trace artifact type is "a label that characterizes those trace artifacts that have the same or similar structure (syntax) and/or purpose (semantics). For example, requirements, design and test cases may be distinct artifact types" [GCHH+12b]. A *trace link* is defined as a "specified association between a pair of artifacts" [GCHH+12b]. This work extends the definition by allowing for more than one source or target artifacts, depending on the trace link type. A trace link can be directed from source artifact to target artifact, otherwise it is undirected. A trace link can be annotated with its trace link type or other semantic attributes. A *trace link type* is "a label that characterizes those trace links that have the same or similar structure (syntax) and/or purpose (semantics). For example, 'implements', 'tests', 'refines' and 'replaces' may be distinct trace link types" [GCHH+12b]. A *trace* consists of the triplet source artifacts, target artifacts, and trace link. Precisely defining RE concepts facilitates to define trace artifacts and trace links as a subset of these concepts; this definition of trace artifacts and trace links directly impacts *tracing* positively. For example, based on the definition of stakeholders and goals, it can be defined to establish a trace link between a stakeholder and its goals. Documenting the refinement links from goals to detailed system requirements may reveal missing requirements, hence improving *external completeness*. Similarly, for cross-references, the referenced entity must exist.

4.6.4 Semi-formal and Formal Representation

The contents of the requirements specification are often documented as unconstrained prose. A textual description of requirements is a commonly agreed specification formalism and understandable by all stakeholders of the system [Sch09]. The down-side is the lack of precision that may lead to quality issues like inconsistencies or ambiguities [Sch09].

A means to add rigor to natural language requirements that is still understandable by all stakeholders is controlled natural language. For example, a series of experiments showed that EARS patterns reduced many common requirements quality issues [MW10]. EARS patterns define a simple sentence structure for system requirements. The patterns improve the *behavioral completeness* of a requirement by reducing missing preconditions, triggers and system responses; they helped to identifying missing requirements, thereby improving *external completeness*; and they reduce vagueness, hence improving *unambiguity*. Controlled natural language that restricts both syntax and semantics may even facilitate an automatic transformation to logical expressions, thereby enabling formal techniques [Sch10].

An experiment compared use case templates with two simple templates to define system interfaces and system functions [MMGD10]. Students used a set of templates to document system requirements for a study object. Students using the case templates documented more interactions of the system with its environment and more information about the required system behavior. The study indicates that using use case templates to document system functions can improve the *behavioral and external completeness* of a requirements specification compared to more simple templates.

Templates and controlled natural language add rigor to natural language requirements, without having the same negative impact on understandability as a graphical or a math-based representation of requirements. As a side effect, a semi-formal representation can also support the *formalization* of an RE concept to a formal representation: Mapping rules from the semi-formal specification technique to the formal technique can guide the formalization, thereby supporting the transition from a textual to a formal representation. If the template defines the information required for a formalization, this increases *activity completeness*.

When a requirement is formalized, it not only facilitates advanced analytical QA techniques; it also directly adds to constructive quality assurance. The formalization of requirements eliminates the *ambiguity* of prose [Bro13a]. A formal modeling language enables a *precise* specification of requirements [Bro13a]. Heimdahl [Hei07] also reports on the constructive benefits of formalizing functional requirements as properties in temporal logics (more precisely, in CTL [CGP99]): "The process of creating a model from the English prose requirements caused us to go back and clarify the English statement of the requirements. In the same way, translating the English statements into SMV [a tool for symbolic model checking[4]] also prompted us to go back and clarify the English statement" [Hei07].

When setting up a formal refinement specification, the stimuli and reactions defined in the involved requirements are mapped. Setting up such a formal refinement specification can identify stimuli or reactions that are missing in the refining or refined

[4] http://www.cs.cmu.edu/~modelcheck/smv.html, lastaccessed9.5.2016

requirements, thereby revealing *behavioral incompleteness*.

Modeling languages and their modeling notations have limitations regarding their expressivity. Typically, not all requirements of a system can be formalized due to a lack of an appropriate modeling notation. For example, real life case studies on formalizing requirements to use cases and contracts using logical expressions could only formalize 79% of the requirements due to the missing support for arithmetic and real-time aspects [NFTJ06]. A classification of requirements into requirement types could indicate which requirement type can be expressed with which modeling notation. Therefore, distinguishing requirements into more fine-grained *RE concepts* can have a positive impact on the formalization. For example, SPES distinguishes requirement types according to different SysML notations [DTW12]. The FOCUS system modeling theory facilitates to formalize *interface requirements*, requirements on the syntactic interface of a system and on its logical interface behavior. AutoFOCUS3 provides a set of specification techniques to model interface requirements. For example, a state automaton is suitable to model the required interface behavior of a system. However, a state automaton is not an appropriate means to model the communication of a system and its environment. AutoFOCUS3 provides message sequence charts that are suitable to model communication (in terms of sequences of messages). For details on FOCUS and AutoFOCUS3 see Section 2.3.

4.6.5 Summary

Creating a requirements specification with the QA quality factors discussed in Section 4.5 can indeed have a positive effect on the system quality factors. In answer to research question RQ 1.4., the table in Figure 4.8 provides an overview of the QA quality factors of the various RE concepts, it lists the activities where these QA quality factors are created and it lists the positive effects on the system quality factors.

		Constructive QA						
		Document entities	Trace entities		Formalize to semi-formal representation	Formalize semi-formal to formal representation	Formalize to formal representation	Formalize to (semi-) formal representation
		System quality factors						
		+ (activity) completeness [RR06, TG14] + type consistency	+ external completeness	+ behavioral completeness	+ behavioral and external completeness [MW10, MMGD10], unambiguity [MW10]	+ activity completeness	+ unambiguity and preciseness [Bro13, Hei07]	+ activity completeness
Artifacts & RE concepts	**QA Quality Factors**	**Impacts**						
Requirements specification	Pre-defined concepts	++ (4.6.1)	+ (4.6.3)					++ (4.6.4)
Concept	Pre-defined attributes	++ (4.6.2)						
Requirement	Semi-formal representation				++ (4.6.4)	+ (4.6.4)		
	Formal representation						++ (4.6.4)	
Trace link	Pre-defined trace artifacts and links		+ (4.6.3)					
	Formal refinement specification			+ (4.6.4)				

Figure 4.8: Summary of the constructive impacts (++, +) of the QA quality factors for automated QA; in brackets behind the impacts are the sections where the impacts are discussed

4.7 Related Work

Quality Models. The quality of a requirements and requirements specifications has been widely discussed in RE. Nonetheless, many approaches [ISO11b, KS98, LSS94] focus on the intrinsic quality of RE artifacts rather than quality with respect to a particular activity. This work instantiates the activity-based quality model (ABREQM) [FMF15] that relates the notion of quality always with activities. In this study, the ABREQM is used to investigate impacts of (target) quality factors on the development of a system and impacts of (QA) quality factors on the quality assurance of a requirements specification. A similar approach to the ABREQM is the Requirements Quality Assessment Framework (RQAF) [TBB+13]. The RQAF facilitates a systematic derivation of quality factors and QA techniques from quality assurance objectives rather than a systematic investigation of impacts. The RQAF does not provide means to explicitly document impacts of quality factors on activities.

Model-based Approaches. The artifact-oriented approach AMDiRE [MF11, p. 142] only briefly discusses the different variations of artifact reference structures or similar artifact-oriented models and their use for quality assurance. SPES [DTW12] also does not discuss quality assurance in the requirements viewpoint, but focuses more on elicitation, specification and communication activities. SPES only presents rules to 'check' the documented contents. Work towards systematic quality assurance was undertaken in the SPES XT project, the successor of SPES, by defining the RQAF that was presented above. AutoRAID [Sch09] investigated the impacts of structuring and formalizing requirements on four the quality factors consistency, unambiguity and on the activities tracing, implementation and verification. The impacts are based on argumentation without pointing to other studies performed in this field. This study incorporates the investigations of AutoRAID, adding empirical evidence when possible. This work integrates these investigations with further empirical and argumentative work on quality assurance.

4.8 Conclusion

Instantiating an activity-based RE quality model for QA facilitates a systematic documentation of impacts of the QA quality factors of the requirements specification on QA activities. To systematically document these impacts, the quality model defines the following elements: Artifacts and RE concepts, QA quality factors, the impact of QA quality factors on QA activities including a rationale, and the QA activities and their impacts on the system quality factors.

In RQ 1.1, system quality factors were identified whose absence in the requirements specification can impact the quality of the system under development. The system quality factors that this study investigated are:

- Adequacy
- Lexical, syntactic, and semantic unambiguity
- External, behavioral, and activity completeness

- Logical and type consistency

The study result of RQ 1.2 is a list of RE concepts whose definition can positively impact QA. For example, validation can be positively impacted by specifying requirement sources or a dictionary. Verification can be impacted positively by trace links that document the implementation of a requirement in a subsequent development artifact. The concepts that this study investigated are:

- Goal
- Use case / scenario
- System requirement
- Refinement link
- External trace link
- Glossary
- Requirement source
- Cross-reference
- QA result

In RQ 1.3, the study investigated QA quality factors that structure and represent RE concepts. These QA quality factors enables model-based QA techniques to support the analysis, validation and verification of requirements. The discussed QA techniques can be conducted more efficient by automation. For an automation of QA, it is necessary to provide an automated access to the contents to be investigated and to accessibly store the results of the QA.

In the course of the study, modeling techniques were discussed that facilitate automated quality assurance. Artifact-orientation provides the means to define an artifact reference structure of the contents to be elaborated during RE. Implementing the artifact reference structure as a data model enables database queries on the data model. These queries can be used for the automated selection of the contents to be investigated and to apply conformance constraints on the requirements specification. Modeling languages provide the means for a semi-formal or formal representation of these contents. Last, but not least, a common system modeling theory for requirements and subsequent development artifacts enables a formal verification.

RQ 1.4 investigated the constructive impacts of the QA quality factors resulting from RQ 1.3 on the quality of the requirements specification. Impacts on system quality factors that emerge during the specification were investigated. QA quality factors already have a positive impact during the specification on unambiguity, completeness and consistency.

As discussed in the limitations in Section 4.2, we do not claim the lists of concepts, impacts and activities to be complete. An unmentioned impact does not imply that there is no impact. Other positive and negative impacts of the concepts on the activities are possible. The concepts, impacts, and activities could be subject to further investigations outside this thesis.

The instantiation of the activity-based RE quality model lead to some interesting findings: 1. Many quality factors from literature would be rather classified as activities or

impacts according to the quality model. A systematic investigation of system quality factors was found in the work of van Lamsweerde [vL09]. Nonetheless, he does not claim to provide an exhaustive list of these quality factors. 2. The activity-based RE quality model manifests the notion of *activity completeness* as one aspect of activity: A requirement or requirements specification is *activity complete* with respect to a specific development activity if it comprises sufficient information to perform that activity.

The study results clarify the impact of QA quality factors of the requirements specification on quality assurance. The study results can be used as an input for developing a model-based RE approach that supports achieving this impact. This thesis develops such a model-based RE approach, the MIRA approach. MIRA consists of three key parts according to the constituents of the quality model. 1.) RE concepts and QA quality factors from the quality model are summarized in an *artifact reference structure* in Chapter 6. 2.) A *guideline* provides guidance for the activities discussed in this study in Chapter 7. The guideline provides step-by-step instructions to instantiate the artifact reference structure and to conduct quality assurance on the artifact reference structure. 3.) The artifact reference structure can be used to derive rules to guide the specification. A second, more strict way to apply the artifact reference structure is to operationalize it as a data model in a *tool*. This forces a user to develop a requirements specification that is conformant to the artifact reference structure. Furthermore, this operationalization enables the automation of quality assurance activities as discussed in the study. The tool is presented in Chapter 8. Figure 4.9 provides an overview how the model-based RE approach relates to the study results. An industrial case study in the train automation domain presented in Chapter 9 demonstrate how to apply MIRA in an industrial context to 4.) meet the discussed system quality factors.

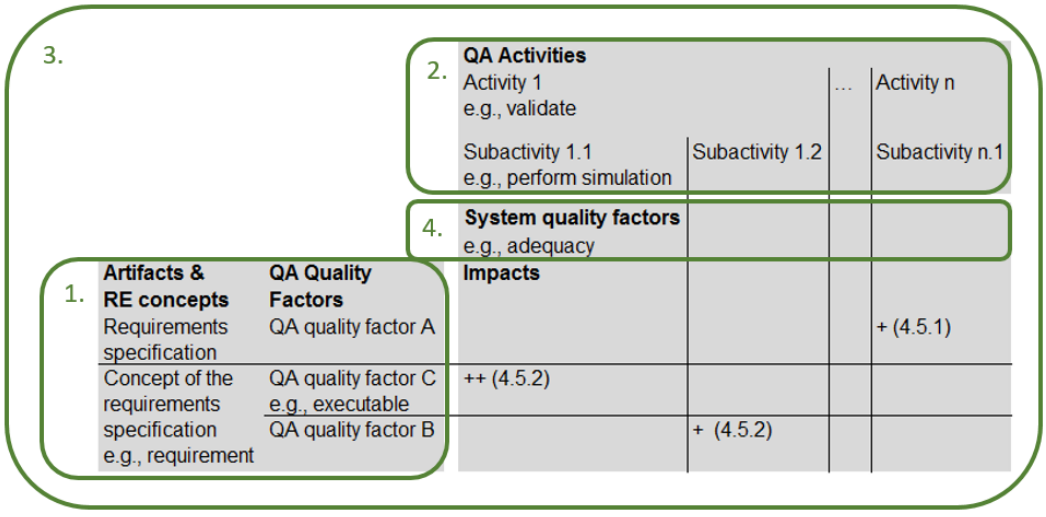

Figure 4.9: MIRA instantiates the 1) RE concepts and QA quality factors in the MIRA artifact reference structure, 2) activities of the study results in the MIRA guideline, and provides 3) an operationalization for the various parts in the MIRA tool; the application in a case study demonstrates how MIRA achieves some of the 4) system quality factors

Chapter 5

Requirements for Model-based RE Tools

This chapter investigates high-level requirements for a model-based RE tool required by the embedded systems industry. In a former study by Sikora et al. [STP12] on the industry needs of RE for embedded systems, practitioners mentioned a lack of tool support for model-based RE. "Experienced practitioners in requirements engineering in the embedded systems (ES) domain agree that models often or always help them understand complex requirements more easily and express a strong wish for using models more intensively in RE. One major obstacle for using models more intensively in RE is the lack of appropriate tool support" [STP12]. The study mentioned some of these requirements and indicated that further requirements for model-based RE tools exist in industry that should be considered in a model-based RE tool. A first search did not reveal a work that systematically investigates the requirements for a model-based RE tool. This lead to the main research question for this chapter:

> ***RQ2:*** *What are the requirements for model-based requirements engineering tools for embedded systems in practice?*

The contribution of this chapter is a two-stage study. In the first stage, we gathered an initial set of requirements for model-based RE tools by a systematic literature review. The needs and challenges described in various papers were transformed into requirements and consolidated into a list of requirements. In the second stage, we validated the relevance of these requirements by expert feedback via a questionnaire. Moreover, we collected additional requirements considered important by practitioners.

The study results were used as an input for the development of the model-based RE approach MIRA in Chapter 6 to Chapter 8.

Contents

5.1 Research Method

The goal of this exploratory study was to obtain a core set of requirements of practitioners that a model-based RE tool for the embedded systems domain should fulfill. Special focus was placed on those requirements that are independent of a concrete modeling language. The requirements should furthermore be independent of a particular development method such as agile development or V-model. Requirements for a concrete language and method could be elicited in a next step based on the results of this work. The requirements have to be specific for model-based RE. This excludes requirements that have a broader scope on model-based development or RE tools. Therefore, requirements for model-based development tools, for instance, concerning the layout of graphical modeling languages, are out of the scope of this study. General requirements for software and system development tools that are neither specific for RE nor for modeling tools, such as version or configuration management, are also out of the scope of this study. The study was conducted by three researchers including the author.

The main research question of this study was

RQ2: What are the requirements for model-based requirements engineering tools for the embedded systems domain in practice?

The study was performed in two parts. Firstly, a systematic literature review was conducted in order to obtain an initial set of requirements stated in literature answering

RQ2.1: What are the requirements of model-based requirements engineering tools for embedded systems in literature?

In order to consolidate and evaluate the list of requirements collected in the first step, we conducted a survey amongst practitioners from the embedded systems domain. This survey was conducted in the form of an online survey. Target participants were practitioners with experience in RE from the embedded systems domain. To assess the participant's context and background, we collected some demographic information such as experience, industrial sector, or company size. We devised the following research questions for experts in the field.

RQ2.2: How do practitioners rate the requirements for model-based requirements engineering tools from literature?

We asked the practitioners to rate the initial set of requirements independently by importance on a 4-point scale from *must-have* to *not relevant* or *unclear* if they did not understand the requirement. The participants were not limited in the number of requirements they can rate as a must-have. To gain further insights into the prioritization of the requirements, each participant was asked to provide the five most important requirements.

RQ2.3: What are additional requirements from practice?

In order to obtain a more complete set of requirements, each practitioner could complement the set of requirements with additional requirements.

5.1.1 Paper Selection and Requirements Extraction

A systematic literature review is a means of identifying, evaluating, and interpreting all the available research that is relevant to a particular research question, topic area, or phenomenon of interest [KC07]. As we had a fairly specific idea of what we were searching for, we conducted a systematic literature review as described in [KC07] to answer the first research question RQ2.1.

Inclusion and Exclusion Criteria. As our starting batch of papers, we investigated the papers in the sources listed in Table 5.1. As sources for relevant literature, we used Google, Google Scholar (G), and scientific conferences (C), journals (J), a symposium (S), and workshops (W) on RE, software and systems engineering and embedded systems. The range of years for the publications was limited to relatively recent publications. The ISO 26262 safety standard was included as an additional source for requirements potentially highly relevant for the embedded systems domain.

Table 5.1: Literature review sources

Type	Name	from	to
G	Google and Google Scholar	–	–
C	IEEE International Requirements Engineering Conference (RE)	2006	2011
C	International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)	2007	2011
C	International Conference on Advanced Information Systems Engineering (CAiSE)	2009	2011
C	International Conference on Software Engineering (ICSE)	2006	2011
C	IEEE Signature Conference on Computers, Software, and Applications (COMPSAC)	2007	2010
C	Embedded Real Time Software and Systems (ERTS^2)	2004	2010
J	Innovations in Systems and Software Engineering	2005	2011
J	Requirements Engineering	2006	2011
J	Software and Systems Modeling	2007	2011
J	IEEE Transactions on Software Engineering	2009	2011
J	Transactions on Software Engineering and Methodology	2007	2011
J	Journal of Systems and Software	2009	2011
S	The Future of Software Engineering (FOSE)	2007	2007
W	Dagstuhl-Workshop: Model-Based Development of Embedded Systems	2006	2011
W	Model-based Methodologies for Pervasive and Embedded Software	2005	2010
ISO	ISO 26262 Road vehicles – Functional safety	2011	–

Originally, only papers that contain the keywords "model-based" AND "requirements engineering" AND "embedded systems" (or synonyms thereof) in the title or abstract should be included. Only few papers included all three keywords. Due to the few hits, we relaxed this initial inclusion criterion. We included papers which handle the tool needs and requirements of model-based RE without limitation to embedded systems. To cover the embedded systems aspect, we also included papers that cover tool needs and requirements for model-based software or systems engineering in embedded systems without special emphasis on RE. We included all papers on problems including open questions that could be solved by a tool, needs from practitioners for a tool, or tool requirements.

From this set of included papers, we excluded papers on existing RE methods, tools and their application. We excluded papers of this group, as the requirements they include are generally at a very detailed level, specific to a concrete solution and based on numerous assumptions. This is in contrast with the high-level requirements identified in this study, which are independent of a particular solution.

Summarizing, the following criteria were applied:

- The paper contains the keywords (“model-based” AND (“requirements engineering” OR “embedded systems”)) or synonyms thereof in the title or abstract
- The paper does not explicitly exclude embedded systems
- The paper describes problems, needs, or tool requirements in RE
- The paper does not present existing RE methods, tools or their application

Resulting Requirements from the Literature Study. The starting batch contained around 3300 papers. We manually analyzed the title and abstract of these papers on the keywords. Based on the keywords, we identified 130 papers for a deeper analysis with respect to the inclusion and exclusion criteria. Finally, we identified 11 sources. From these 11 sources, we extracted 46 statements that constitute requirements, challenges, and problems in scope, transformed them into requirements and then consolidated them into 23 requirements. Most of the statements we collected were phrased as a description of problems or needs. We rephrased these statements into requirements. Where a reformulation as a requirement was not possible, the original statement was omitted. Due to their diverse sources, the collected statements did not use homogeneous wording. Thus, we harmonized the statements by introducing a common terminology, consolidating similar requirements, deleting duplicates and finally reformulating them into concise statements. This last step was taken to prevent longer statements causing disinterest in the survey. The 11 sources and the resulting list of 23 requirements for model-based RE tools is presented in Table 5.2 and Table 5.3. We used these requirements as the input for the survey with practitioners in industry.

5.1.2 Participants Selection

The target group for the questionnaire was experts from industry, preferably with high experience in RE and model-based software engineering. The population is large enough to warrant the use of a sample for our study. We invited around 100

Table 5.2: The requirements for model-based RE tools and their sources in literature (1/2)

Source	Requirement
ISO 26262 [ISO11a]	Support the whole life-cycle of an embedded system inclusive change management
	Providing mechanisms for documenting the results of validation and verification tests
	Support the specification of requirements in such a way that the test cases can be derived (semi-)automatically
	Support the representation and propagation of software criticality levels (e.g., SIL2, ASIL-C)
Sikora et al. [STP11]	Support for different, well-defined system abstraction layers (product, system function and component views)
	Support refinement, traceability, and consistency checking across different abstraction layers
	High level of automation in creating and maintaining traceability links between RE and design
Matulevičius and Strašunskas [MS03]	Support different representation forms of requirements: informal, semi-formal like UML, formal (mathematics-based)
	Support document-generation
Egyed et al. [EGHB07]	Support of tracing activities: creation, utilization (e.g., support change impact analysis), maintenance, enhancement
	Allow different tracing granularities (within requirements, between requirements and design)

international experts in software and systems engineering from our contact lists. We chose participants matching our target criteria, with a high affinity to RE and model-based software engineering to increase the quality of the survey results and our confidence in them independently of the number of respondents. The research method and first results were summarized in a technical report [TKM13] that was also sent to all participants who provided their email.

5.1.3 Feedback Elicitation

We implemented our survey in the form of an online questionnaire, based on the guidelines described in [Bra08], consisting of the two blocks *participant information* and *tool requirements*.

Participant information included the participants' country, company size, sector and working experience. Their working experience in years, working experience in RE in years, and the level of involvement with RE were mandatory questions. The following answers to the level of involvement were possible: that RE is one's *main task*, that the respondent is *involved, but not as a main task* or that it is *part of everyday work, but not directly involved*. The respondents were also asked to list the models and modeling

Table 5.3: The requirements for model-based RE tools and their sources in literature (2/2)

Source	Requirement
Pretschner et al. [PBKS07]	Support the representation of non-functional requirements (safety, security, reliability, maintainability, etc.)
	Support platform-specific legacy constraints
	Support the creation and reuse of requirements with varying levels of detail, e.g., requirements to reuse ECUs, reliability, maintainability, etc.
	Support variant management
Berenbach [Ber10]	High level of automation in creating and maintaining traceability links within requirements
Berenbach and Borotto [BB06]	Support requirements metrics, e.g., quality metrics like use case completeness
Fabbrini et al. [FFLS08]	Support for the integration of tools and their artifacts (for example domain specific tools integrated by import/export interfaces)
	Deep integration of the model-based RE tool in an existing development process (direct support for interactions between activities, consistency checks)
Streitferdt et al. [SWN+08]	Context specific tool tailoring, e.g., through UML profiles, new plug-ins, ...
Heimdahl [Hei07]	Support high level of automation of the validation and verification of requirements
	Support model validation through tight integration of requirements and design
Schmid et al. [SRB+00]	Support simulation of (executable) requirements

languages they have already used in RE. This answer further evaluates the experience of the participants regarding the research question. For additional background information, the participants were asked whether they are involved in safety-critical product development.

Tool requirements: this block contains all 23 requirements identified in the literature review and an ordinal scale [FP99] to evaluate them. The scale is rated with the following, self-explanatory options, as an answer to research question *RQ2.2*:

- *Unclear requirement*, hereafter shortly referenced as *unclear*
- *Not relevant*
- *Nice to have, but not necessary*, hereafter shortly referenced as *nice to have*
- *Important, but could live without it*, hereafter shortly referenced as *important*
- *Must-have*

To prioritize the requirements, the respondents were asked to select the five highest priority requirements in their opinion. Finally, in a free text form, we asked for

additional requirements and comments to cover *RQ2.3*.

5.1.4 Data Analysis

The participants' data was filtered by exclusion criteria. To focus on our target group, we excluded responses from participants for whom RE is *part of everyday work, but* they are *not directly involved*. The information on the working sector was used to focus on responses from participants with practical industrial experience by excluding participants from the *applied research* and *education* sectors.

We evaluated the participants' rating. For each requirement, we counted the number of answers for each option. For each requirement, we furthermore calculated the percentage of answers for each option. For each requirement, we counted the number of participants who included this requirement in their priority list. All requirements in the priority list of a participant were treated equally in the evaluations, for example, a list containing requirements D1, D2 and D3 is equivalent to a list containing requirements D2, D3, D7. The number of *unclear* requirements indicates whether practitioners understood the requirements. Furthermore, we analyzed whether importance rating and prioritization correlate, or whether there is a deviation. We extracted the most important and most prioritized requirements to give an overview of the most highly rated requirements. Finally, we checked, how many participants considered a requirement as a *must-have* or *important*. If the majority of participants considers a requirement at least important, we see this as an indication that the requirement should be considered in the development of a model-based RE tool.

The additional requirements were categorized into requirements in scope of the survey (language and method independent requirements for a model-based RE tool) and requirements not in scope.

Finally, we consolidated the initial requirements with the additional requirements stated by the practitioners.

5.2 Study Results

The first result of the study was that we did not identify a paper with a concise list of requirements for model-based RE tools in existing literature. As described in Section 5.1, the 11 papers in scope of the study resulted in a list of 23 requirements that were to be rated by practitioners in a questionnaire.

5.2.1 Sample Demographics

The study received answers from 29 participants. From these, the answers of participants without industrial experience were excluded; the remaining 22 answers were filtered by desired criteria regarding experience in RE and industry relevance. The demographics of the 22 participants are summarized in Table 5.4. The majority of the participants came from Germany. The main sector with 12 participants is the automotive industry. There was no predominant company size or a clear inclination towards safety-critical products in the demographics.

Table 5.4: The demographics of the participants

Criterion	No. of answers	Results
Working experience	22	Between 5 and 29 years, average: 17 years
Working experience in RE	22	Between 1 and 25 years, average: 9.4 years
Countries	22	Germany: 16, France: 3, U.S.A: 1, Austria: 1, UK: 1
Company size	20	10 participants from small and medium sized enterprises with up to 250 employees, 10 persons from large companies with at least 1,000 employees
Sectors	22	Automotive: 12, consulting: 3, systems engineering: 2, aerospace: 2, aeronautics: 1, telecommunications: 1, information technology: 1
Safety-critical products	21	12 persons work on safety-critical products, 9 persons on non-safety-critical products

19 participants provided information about the modeling languages they already used for RE. These participants gave a list of the different models and modeling languages. Most common languages are SysML mentioned by 8 persons, UML with 7, MATLAB/Simulink with 6 and EAST-ADL with 4 persons. The participants stated several other languages including BPMN, Event-B, or domain-specific-languages; each of these languages was only stated by one person.

5.2.2 Rating of the Requirements by Practitioners

22 participants rated all 23 requirements, so in total 506 ratings have been provided. In the questionnaire, we asked the participants to prioritize five requirements. 19 participants provided a priority list of requirements that included between one and seven requirements, on average each participant prioritized 4.5 requirements. An overview of the requirements with the number of participants rating each option and the number of participants that included the requirement into the prioritized list is given in Table 5.5 and Table 5.6.

The brief description of requirements entails the risk that the practitioners are not able to understand a requirement. In 12 cases, a participant rated a requirement as *unclear*. The maximum number of *unclear* ratings for one requirement is 3. Due to this relatively low number, we do not believe that this has significantly influenced our results. Nevertheless, we included this threat into our threats analysis in Section 5.4.

We investigated whether the prioritization of the requirements and their rating correlate. Practitioners selected 98 times a requirement in their prioritization list. In 88% of the cases, a practitioner that prioritized a requirement also considered it a *must-have*. 9% of the prioritized requirement are also considered *important*. 3% of the prioritized requirements are rated as *nice-to-have*. The distribution indicates a correlation between rated importance and appearance on the priority list.

Table 5.5: Number of ratings as must-have (1), important (2), nice to have (3), not relevant (4) and unclear (?) and prioritizations (P) for each requirement (1/2)

ID	1	2	3	4	?	P	Requirement
D1	14	7	1	0	0	7	Support the representation of non-functional requirements (safety, security, reliability, maintainability, etc.)
D2	6	5	5	3	3	2	Support platform-specific legacy constraints
D3	15	4	2	0	1	3	Support the whole life-cycle of an embedded system inclusive change management
D4	12	5	3	0	2	8	Support different representation forms of requirements: informal, semi-formal like UML, formal (mathematics-based)
D5	20	2	0	0	0	8	Support document-generation
D6	11	10	1	0	0	5	Support for different, well-defined system abstraction layers (product, system function and component views)
D7	15	7	0	0	0	3	Support refinement, traceability, and consistency checking across different abstraction layers
D8	7	7	6	0	2	2	Support the creation and reuse of requirements with varying levels of detail, e.g., requirements to reuse ECUs, reliability, maintainability, etc.
D9	12	8	2	0	0	7	High level of automation in creating and maintaining traceability links within requirements
D10	13	6	3	0	0	5	High level of automation in creating and maintaining traceability links between RE and design
D11	11	7	3	0	1	4	Support of tracing activities: creation, utilization (e.g., support change impact analysis), maintenance, enhancement
D12	6	7	6	1	2	1	Allow different tracing granularities (within requirements, between requirements and design)

The following requirements were ranked by the most participants as a *must-have*, see also Figure 5.1:

1. (D5) document generation
2. (D7) support of refinement, traceability and consistency checking across different abstraction levels
3. (D3) support the whole life cycle of an embedded system including change management
4. (D1) support the representation of non-functional requirements (safety, security, reliability, maintainability, etc.)
5. (D13) support for the integration of tools and their artifacts (for example domain specific tools integrated by import/export interfaces)

Table 5.6: Number of ratings as must-have (1), important (2), nice to have (3), not relevant (4) and unclear (?) and prioritizations (P) for each requirement (2/2)

ID	1	2	3	4	?	P	Requirement
D13	14	6	2	0	0	5	Support for the integration of tools and their artifacts (for example domain specific tools integrated by import/export interfaces)
D14	8	8	4	0	2	2	Deep integration of the model-based RE tool in an existing development process (direct support for interactions between activities, consistency checks)
D15	6	6	4	3	3	2	Context specific tool tailoring, e.g., through UML profiles, new plug-ins, ...
D16	7	10	3	1	1	4	Support high level of automation of the validation and verification of requirements
D17	7	8	6	0	1	2	Support model validation through tight integration of requirements and design
D18	11	6	5	0	0	3	Providing mechanisms for documenting the results of validation and verification tests
D19	5	6	8	3	0	1	Support simulation of (executable) requirements
D20	6	11	5	0	0	3	Support the specification of requirements in such a way that the test cases can be derived (semi-) automatically
D21	7	7	7	1	0	0	Support requirements metrics, e.g., quality metrics like use case completeness
D22	9	7	3	1	2	2	Support the representation and propagation of software criticality levels (e.g., SIL2, ASIL-C)
D23	10	7	3	0	2	3	Support variant management

The following requirements were prioritized by the most participants, see also Figure 5.2:

1. (D4) different representation forms of requirements: informal (flowing text description), semi-formal (UML diagrams), formal (mathematics-based)
2. (D5) document generation
3. (D1) support the representation of non-functional requirements (safety, security, reliability, maintainability, etc.)
4. (D9) a high level of automation in creating and maintaining traceability links within requirements

Between 50% and 100% of the participants rated each requirement *important* or *must-have*. This percentage is even higher for participants working on safety-critical products, with 59% of the participants considering all of the requirements *important* or a *must-have*. We conclude that all requirements identified in the literature review can be considered relevant by practitioners and therefore have to be addressed when developing a model-based RE tool.

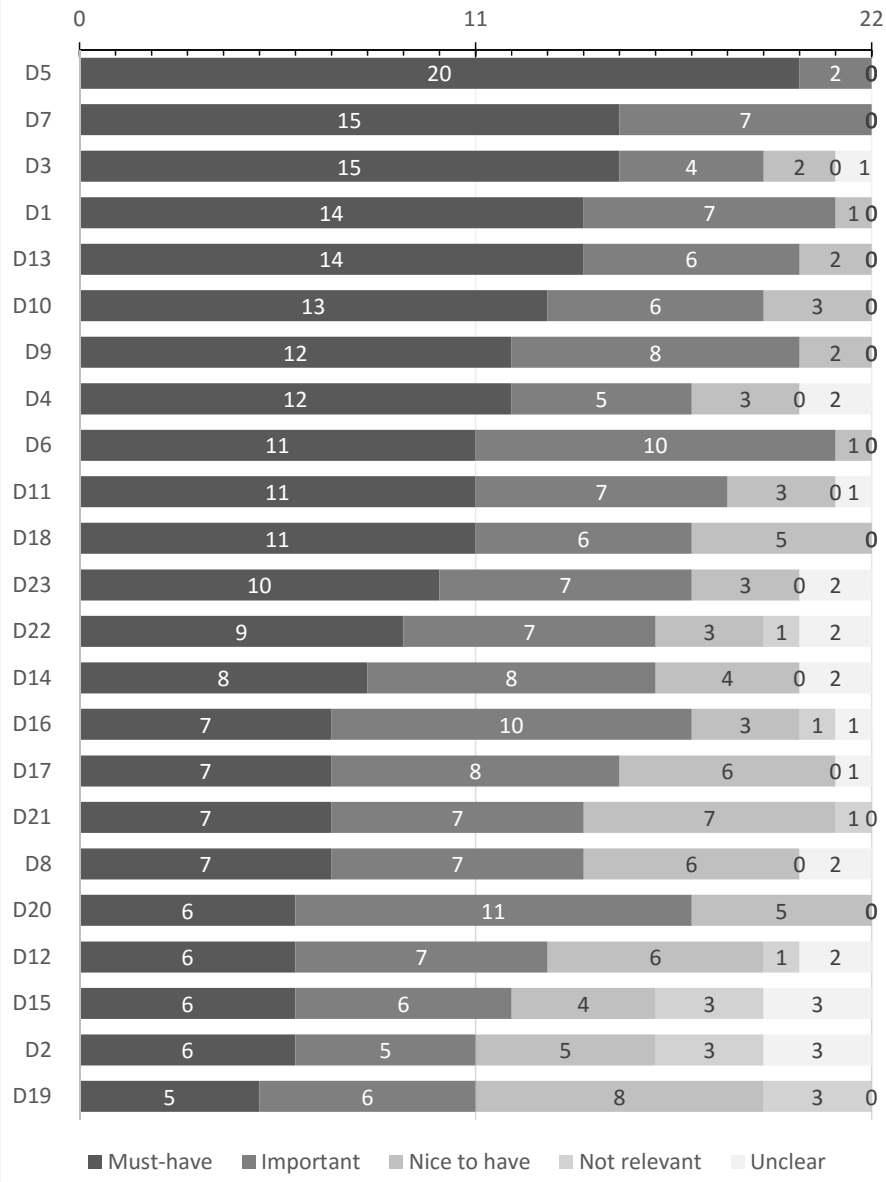

Figure 5.1: Bar chart of the rating of the requirements sorted by number of ratings as *must have*

5.2.3 Additional Requirements from Participants

10 participants stated 27 additional requirements from which 14 requirements are in scope of the survey. Requirements for concrete modeling languages, for roles and responsibilities of the resulting specification, for example *interfacing and requirement handshake features from customer to supplier*, and extra-functional requirements such as *easy to use* or *well affordable* have been excluded as they are not specific for a model-based RE tool and hence not in the scope of this study. The resulting additional requirements in scope of our study are listed in Table 5.7.

Two of the additional requirements are refinements of D1, the requirement for support of different representation forms: A1 requires the provision of prose as addi-

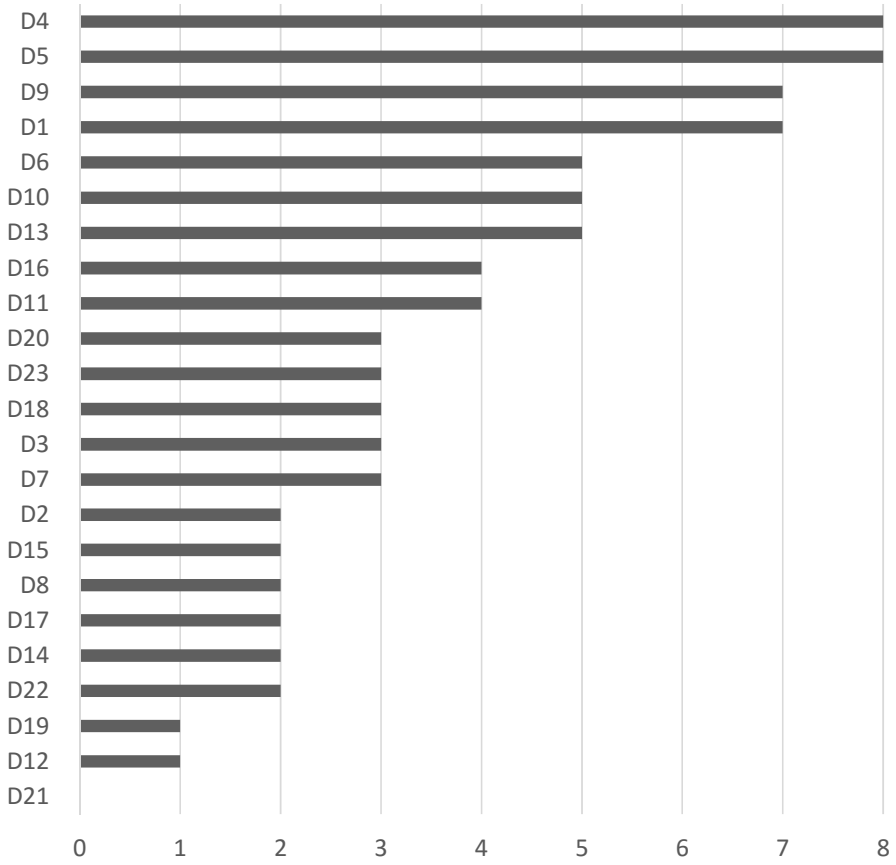

Figure 5.2: Bar chart of the number of prioritizations of each requirement

tional information to models; A3 requires that models should be directly created in the tool. In A2, a practitioner argues that patterns to control natural language are more appropriate than requirement models when communicating with customers, as patterns would reduce the risk of misunderstandings at a very early stage. He also suggests that requirement models are more appropriate to communicate with developers. Therefore, there is a need for correlating different notations for different stakeholders. Requirement A4 indicates the need to support the elicitation workflow. This additional requirement could also be partly addressed by offering different representation forms within the tool (D1); prose for brainstorming, a more formal representation form for precise requirements.

Practitioners provided additional requirements for tracing. Requirement A5 emphasizes the need to predefine and check trace links. Requirement A6 emphasizes the traceability between requirements and models. A7 emphasizes the need to trace requirements and verification and validation artifacts.

Another prominent topic was the compatibility with other tools. Although tool integration was mentioned in the original list (D13), support for data import in A8 and specific interchange formats, especially ReqIF, was required in the additional requirements A9, A10 and A11. In A13, a practitioner explicitly identified the need for structured guidance for data input. Requirement A14 also aims at the support of documenting requirements.

Table 5.7: Additional requirements for model-based RE tools as stated by practitioners

ID	Requirement
A1	Combination of model based and "natural language" requirements as additional information
A2	Ability to create models directly in the tool
A3	Support by text shapes (compare CPRE-Foundation Level or Sophists REgelwerk). There need to be support in the documentation of requirements between requirement analyst (RA) and the customer. Model based requirements tend to support the interface between RA and development. When the requirements of "interface 1" are misunderstood, you going to "translate" them also misunderstand with the model based language.
A4	Support requirements elicitation workflow, i.e. from brainstorming up to hierarchical, precise documented requirements
A5	Providing mechanisms for defining and verifying the traceability data model between expected artifacts (e.g., a traceability rule checker)
A6	Ability to trace components of a model to requirements (e.g., a step in a process flow)
A7	Creating and maintaining traceability links between requirements and verification and validation artifacts
A8	Ability to import data from other tools
A9	Support of "ReqIF"
A10	Support for standard model and requirement interchange formats (e.g., ReqIF, XMI)
A11	Support standard (ReqIF) for interchange Requirement Format
A12	Flexibility in generating documents and reports
A13	Structured guidance for data input
A14	Provide text highlighting of keywords, help in formulation like text proposals and automatic correction. I.e. usability during the documentation step

5.2.4 Consolidation of the Requirements

The requirements identified in this study have been rephrased in order to match the vocabulary used in the remainder of this work and consolidated with the additional requirements from the practitioners. Thereby, the requirements have been grouped into topics. Table 5.8 shows the mapping of the draft and additional requirements to the consolidated requirements. The resulting set of requirements yielded 31 language- and method-independent requirements for a model-based RE tool. An overview of all requirements is given in Table 5.9 to Table 5.11.

Table 5.8: Mapping of draft and additional requirements to the consolidated requirements

Draft and Additional Requirement IDs	ID Consolidated Requirement
D4, A2, A4	R1
A1	R2
A3	R3
D11	R4
A5	R5
A6	R6
D12	R7
D9	R8
D10	R9
A7	R10
D5, A12	R11
D13, A8	R12
A9, A10, A11	R13
D16	R14
D17	R15
D20	R16
D19	R17
D18	R18
D21	R19
D6	R20
D7	R21
A13	R22
A14	R23
D14	R24
D3	R25
D1	R26
D2	R27
D23	R28
D15	R29
D22	R30
D8	R31

Table 5.9: The consolidated requirements for model-based RE tools grouped by topics (1/3)

ID	Requirement
Representation Forms	
R1	The tool should support the representation of requirements as unconstrained prose, semi-formally (UML-based or as constrained natural language) and using a formal (math-based) modeling language.
R2	The tool shall facilitate the documentation of a natural language description for requirement models in order to give additional information to the models.
R3	Models should be created directly in the tool.
Tracing	
R4	The tool shall facilitate the creation, utilization (e.g. support change impact analysis), maintenance, enhancement of trace links.
R5	The tool shall provide mechanisms for defining and analyzing trace links between expected artifacts such as a traceability rule checker.
R6	The tool shall facilitate to trace model elements to requirements
R7	The tool shall enable to document traces at different tracing granularities (within requirements, between requirements and design).
R8	The tool shall provide a high level of automation in creating and maintaining trace links within requirements.
R9	The tool shall provide a high level of automation in creating and maintaining trace links between RE and design.
R10	The tool shall facilitate to create and maintain trace links between requirements and verification and validation artifacts.
Interfaces	
R11	The tool shall facilitate to generate documents.
R12	The tool shall be capable to be integrated with other tools and their artifacts, for example by import/export interfaces.
R13	The tool shall support standard requirement interchange formats such as ReqIF.

5.3 Discussion: Tool Support for Quality Assurance

The requirements for model-based RE tools contain concrete quality assurance activities as well as entities and their characteristics with concrete impacts on quality assurance. Thereby, the study indicates that quality assurance is an important issue for model-based RE tools. The requirements concretize the activity-based RE quality model elaborated in Chapter 4.

Table 5.10: The consolidated requirements for model-based RE tools grouped by topics (2/3)

ID	Requirement
Quality Assurance	
R14	The tool shall provide a high level of automation of the validation and verification of requirements
R15	The tool shall support the verification of the system design through tight integration of requirements and design.
R16	Requirements shall be specified in such a way that the test cases can be derived (semi-)automatically
R17	The tool shall provide a simulation of (executable) requirements
R18	The results of validation and verification tests shall be documented in the tool.
R19	The tool shall calculate and display requirements metrics, for example to measure use case completeness.
Abstraction Layers	
R20	The tool shall provide different, well-defined system abstraction layers for the system.
R21	The tool shall facilitate to document refinement links and other trace links as well as consistency checking across different abstraction layers
Guidance	
R22	The tool shall provide a structured guidance for the data input.
R23	The tool shall address the usability of the documentation by providing capabilities such as highlighting keywords in text, help in the formulation like text proposals and automatic correction.
R24	The tool shall be integrated in an existing development process.
R25	The tool shall support the whole life-cycle of an embedded system inclusive change management

5.3.1 Analytical Quality Assurance

The list of requirements includes a set of analytical quality assurance activities that the tool should support and a set of objects that should be subject to quality assurance. These are requirements (R14, R16, R17, R19), trace links (R5) including refinement links (R21), and system design (R15). The quality assurance activities include the analysis (R5, R17, R21), validation (R14, R16) and verification (R14, R15) of these concepts. A model-based RE tool should support quality assurance activities by automation (R14, R16, R17, R19). The tool should furthermore be able to capture quality assurance results (R18).

The study showed the need for a model-based RE tool to exchange information with its environment through interfaces. These interfaces include document generation

Table 5.11: The consolidated requirements for model-based RE tools grouped by topics (3/3)

ID	Requirement
Requirement Types	
R26	The tool shall support the documentation of extra-functional requirements.
R27	The tool shall support the documentation of requirements for platform-specific legacy constraints
Others	
R28	The tool shall support variant management.
R29	The tool shall facilitate a context specific tailoring of the tool, for example through UML profiles or new plug-ins.
R30	The tool shall support the documentation and propagation of software criticality levels, for example the safety integrity levels SIL 1 to SIL 4.
R31	The tool shall support the creation and reuse of requirements with varying levels of detail.

(R11) and interfaces to other tools (R12, R13). These interfaces can also become relevant in the validation and verification of the requirements specification. A manual validation and verification requires the access of the stakeholders to the documented information; document generation is a means to achieve this access. An automated verification requires an automated connection of requirements and the verification object.

5.3.2 Constructive Quality Assurance

Representation Forms. The study confirmed that practitioners demand a tool that supports prose as well as a formalization of requirements to a semi-formal and a formal representation (R1, R2, R3). The study in Chapter 4 discussed the impact of representation forms on quality assurance.

Tracing. The study yielded a set of requirements for the specification and (automated) evaluation of trace links (R4, R5, R6, R7, R8, R9, R10). A practitioner pointed out the need for clearly defined trace rules that can be checked by a trace rule checker (R5). Some practitioners also demanded that links are not only relevant between requirements or requirements and design, but also from a requirement documented as prose to its semi-formal representation in a model (R6), and to verification and validation artifacts (R10).

Abstraction Layers. The study showed the need to define different abstraction layers for the system (R20), for example for the entire system or only parts of the system. These abstraction layers are required in order to document and check (R21) the refinement across these abstraction layers.

5.3.3 Guidance

Practitioners stated requirements on the guidance to be provided by a model-based RE tool: Practitioners highlighted the need to guide the tool users in the application of the tool (R22). This requirement implies that also the constructive and analytical quality assurance should be guided. A means to provide constructive support in a tool are keyword highlighting, text proposals and automatic correction (R23). An integration in an existing development process (R24) supports the verification of requirements against subsequent development artifacts.

5.3.4 Requirements without Impact on Quality Assurance

Requirement Types. The study did not specifically emphasize the need to document functional requirements, presumably as the support for these requirements is so fundamental that it seemed obvious to the participants. The study showed the need to document extra-functional requirements (R26, R27). The requirements do not discuss an impact of distinguishing these requirement types on quality assurance.

Others. Further requirements without a direct impact on the quality assurance have been provided. These comprise the support of variant management (R28), a context-specific tailoring of the tool (R29), the support for safety criticality levels (R30) and the support for reuse of requirements (R31).

5.4 Threats to Validity and Limitations

The study faces the following threats to internal and external validity.

Internal Validity. The choice and design of a research method influences the quality of the result. The quality of a systematic literature review depends highly on the search parameters. A systematic literature review can only identify the publications that are available in the data sources that are investigated. Therefore, this study includes the most renown journals and conferences in RE and software engineering as data sources, for example, the Requirements Engineering journal published at Springer, or the IEEE Requirements Engineering Conference. Adding further sources might increase the number of publications identified in the search at the risk of reducing the quality of the papers.

The search string can crucially influence the results of the literature research. Additional search terms might increase the list of resulting papers. Therefore, we complemented our search by a manual investigation, taking into account potential synonyms used by the authors.

The exclusion criteria applied in the review might lead to missing requirements, that could be relevant to practitioners. Furthermore, authors of these papers stem not only from industry, but also from academia. Practitioners might have a different perspective on the requirements for a model-based RE tool than researchers. We

addressed this threat by combining the systematic literature review with a survey amongst practitioners.

Extracting requirements from the publications is a highly subjective task. Different researchers may extract different requirements. By reviewing the extraction process by a second researcher we mitigated this threat.

The systematic literature review resulted in a set of requirements with a significantly different level of detail. Different levels of detail make it more difficult to rank and compare the requirements. We had two options to process these requirements. We could align these requirements by interpreting and refining them. These changes would have inferred new threats. We chose to accept the different levels of detail, thereby accepting a limitation for the users of the study results.

The quality of a survey via a questionnaire is influenced by the design of the questionnaire. We chose to limit the questionnaire: Practitioners could rate every requirement according to their relevance by a predefined set of values. In order to make sure this limitation did not lead to an oversight the participants could also add missing requirements. The results might be influenced by the layout and order of questions in the questionnaire.

All requirements have been ranked as *important* or *must-have* by at least 50 percent of the participants. This could also be seen as a threat to validity as preferences of participants are not visible at this level of detail. This threat was partly mitigated by a) providing a fine-grained scale, where important requirements are distinguished from must-haves and b) including a prioritization of the requirements.

Any questionnaire always carries the risk of misunderstandings. The respondents may have a different understanding of the requirements and the terms used in the requirements. This may lead to a false ranking of the requirements. Therefore, we tried to use a homogeneous set of commonly understood terms throughout the entire survey. To further mitigate this risk, requirements could be rated by the respondents as *unclear*. The number of participants that rated a requirement as unclear is listed in Table 5.5 and Table 5.6; unclear requirements indicate that, for further processing, these requirements should be reassessed. Nevertheless, none of the requirements were considered as unclear by the majority of the participants. The same risk of misunderstandings also relates to the additional requirements that were stated by the participants. This threat could not be resolved, but by providing the literal requirements in Table 5.7, we tried to make this transparent to the reader.

External validity. Self-administered surveys are generally threatened by low responses and response rates [Bra08], which leads to a statistically insignificant number of results. The number of participants directly influences the generalizability. We chose to target a group of participants with high experience in RE and model-based software engineering to increase the quality of the survey results and the confidence in them, independent of the number of respondents. Only participants with a high affinity to the target topics and a high experience in the subject of the study were chosen. By achieving a high quality of the responses, we intended to compensate the low number of responses.

Most, but not all of the respondents are located in Germany, which may threaten the

transferability to other countries. However, many of these German respondents work in companies acting globally with experience in internationally distributed projects. Finally, the various company sizes of the respondents may affect the results, as the RE needs and activities of small and medium enterprises may be different from those of large companies with well-established processes.

Limitations. The requirements resulting from this study are high-level stakeholder goals. The requirements have a varying level of detail as they are kept on the level of detail that they were extracted from the literature and as they have been given by the practitioners. These requirements catch essential characteristics a model-based RE tool should have, but they are not detailed system requirements that can be directly implemented in a model-based RE tool. To obtain a detailed, atomic, and measurable system requirements specification would require to break down these requirements further and to tailor them to a specific context and to specific modeling languages and their capabilities and constraints. Therefore, we recommend the users of the study results to perform a refinement of the requirements to the level of detail necessary for their purpose. Another reason might infer the need to revise the list of requirements. Some tool user groups may have very specific tool requirements, for instance, opposed by standards and regulations. Therefore, we recommend tool vendors to perform an additional survey of practitioners from specific target groups to complement the list of requirements.

5.5 Related Work

Roadmaps and Challenges. Several academic studies exist enumerating the many challenges and problems facing model-based software development in general, as well as model-based RE, and providing visions for improvement. They often do not discuss tools or tool requirements for model-based RE in depth. Cheng and Atlee [CA07] discuss challenges and research directions in RE, including modeling activities, but without any focus on tools. Nuseibeh and Easterbrook [NE00] present an overview of modeling in RE, but aside from presenting requirements management tools they do not discuss any tool-specific topics further. France and Rumpe [FR07] provide a research roadmap on model-driven development of complex software, but without focus on RE. Broy [Bro06] discusses challenges in automotive software engineering, pointing out the importance of tools in model-based development, also in RE. Heimdahl [Hei07] points out challenges in safety for software-intensive systems like medical devices, which were extracted and transformed into requirements. Domain-specific analysis of challenges, requirements, or key activities in model-based RE are presented in Streitferdt et al. [SWN+08] and Fabbrini et al. [FFLS08]. Pretschner et al. [PBKS07] provide an overview of current challenges of model-based RE in their roadmap on software engineering for the automotive domain.

The roadmap of Gotel et al. [GCHH+12a] from 2012 investigates software and systems traceability research, thereby presenting and discussing potential solutions for the model-based RE tool requirements related to tracing presented in this chapter, including solutions for trace data models (A5), automated tracing (D9, D10) and handling different tracing granularities (D12). In a case study at six companies from 2014,

Bjarnson et al. [BRB+14] identified challenges and practices in aligning requirements with verification and validation; the challenges mentioned by industry experts confirmed the tracing requirements A7 (tracing between requirements and validation and verification artifacts) and R7 (tracing across abstraction layers, here called abstraction levels).

Needs and Requirements. The standard ISO/IEC TR 24766:2009 [ISO09] gives general tool requirements for RE without addressing the challenges of model-based development. The recommendation Z.150 [ITU11b] specifies the language requirements for a user requirements notation in the field of telecommunications. Matulevičius and Strašunskas [MS03] present an evaluation framework for verification and validation that contains requirements in scope of this study. Sikora et al. [STP11, STP12] conducted an industrial study with 17 representatives from large, internationally operating companies in the domain of embedded systems in Germany on industry needs for RE, including model-based RE. This work is not specifically focused on requirements for tools but contains needs for tools and thereby was an important source for our initial set of requirements. Domain-specific standards such as the DO-178C standard [DO112] for avionics and the ISO 26262 automotive safety standard [ISO11a] contain domain-bound, implicit and disjoint requirements for RE activities and development tools.

In the work from 2013, Hesari et al. [HBY13] investigate industrial challenges and needs for requirement-based test generation, emphasizing the need for an integrated computer-based solution for reusing test artifacts; some of the specific challenges for requirement-based test generation such as tracing to test artifacts and variability management also have been identified in other papers as challenges and needs for model-based RE tools.

Tool Capabilities. The ISO/IEC TR 24766:2009 standard [ISO09] provides 157 tool capabilities that can be used as evaluation criteria for RE tools. This standard provides an overview of potential features of a tool, including features for the model-based specification of requirements. Schmid et al. [SRB+00] compares tools and their capabilities with respect to simulation. Matulevičius and Strašunskas [MS03] present an evaluation framework for RE tools with respect to verification and validation, including but not limited to model-based RE.

In 2012, Gea et al. [CDGNFA+12] assessed RE tools based on the capabilities described in the ISO/IEC TR 24766:2009 standard. They report that current RE tools support best capabilities for the elicitation of requirements whereas requirements modeling and management are the least supported group of capabilities. The capabilities have not yet been rated with respect to their importance for users in the field of model-based RE. A next step would be to align the capabilities with user requirements for model-based RE tools.

We could find no source that holistically presents, discusses, or evaluates a collection of requirements for a practical model-based RE tool of embedded systems. We found very few sources that target almost all focus points such as RE tool requirements.

5.6 Conclusion

This study investigated language- and method-independent requirements for a model-based RE tool. The study included a literature review and a questionnaire amongst practitioners. The literature review showed that only a few related studies in this field exist that were incorporated in this work. The literature review revealed 23 draft requirements that were used as an input for a survey amongst practitioners working in the field of RE. The result of the survey in industry was a rating of these requirements for importance and a set of additional draft requirements from practitioners. 22 practitioners participated in the study. The number of participants may not be representative, but is comparable with similar studies in this field such as [STP11]. The participants' high average experience of 9.4 years indicates that the study results are trustworthy. Each of the 23 draft requirements was rated as a must-have or important by at least 50% of the practitioners. The practitioners stated 14 additional requirements. The draft requirements and the additional requirements have been consolidated in a core set of 31 requirements for model-based RE tools.

The survey confirmed and detailed the four challenges that motivate the MIRA approach, see Section 1.2. The study results indicate that practitioners have a demand for a tool that supports heterogeneous representation forms of requirements (R1 – R3), that provides support for the quality assurance of requirements (R14 – R19), and that is tightly embedded in the system development (R7, R9, R15, R20, R21, R24). The study results also indicate that practitioners not only require a simple modeling tool that facilitates the application of a modeling language such as UML. The tool should provide guidance (R22 – R25), for example, for the data input and for the process from brainstorming to precisely formulated requirements.

The requirements have been considered in the development of the MIRA approach in Chapter 6 to Chapter 8. Some requirements had to be excluded from the MIRA approach as they a) are not concerning the quality assurance of requirements and b) are too high-level and treat extensive and advanced topics. The requirements that are excluded concern variability (R28), a systematic reuse of requirements (R31) and change management (R4, R25). Only after the basics of model driven development have been comprehensively worked out, these additional dimensions can be investigated.

Further similar studies in industry on requirements for model-based RE approaches would facilitate the understanding of actual industrial needs. These studies could for example investigate more concrete requirements for a model-based RE tool from industry based on the results of this study.

Chapter 6

The MIRA Artifact Reference Structure

An artifact reference structure is a blueprint of the contents to be elaborated during RE. Instantiating the artifact reference structure yields a requirements specification. An artifact reference structure defines a set of RE content items, concepts, their attributes, representation forms and dependencies. An artifact reference structure can be implemented in a set of rules that guide the specification. Additionally, the artifact reference structure can be implemented as a data model in a tool to restrict the specification.

The research question that drove the development of the MIRA artifact reference structure is:

> ***RQ3:*** *What is a minimal, but extensible artifact reference structure for model-based RE that supports the effective and efficient model-based quality assurance of functional requirements?*

The contribution of this chapter is the MIRA artifact reference structure, where each concept, attribute, representation form and dependency has an impact on the effectiveness or efficiency of model-based quality assurance of functional requirements. These impacts have been investigated in a study in Chapter 4. The MIRA artifact reference structure is developed constructively by integrating the concepts and QA quality factors from this study. To enable the verification of functional requirements, the MIRA artifact reference structure is embedded in a modeling framework for software and systems engineering, see Section 2.3.4. However, these integrations highlighted some gaps in the artifact reference structure. Therefore, in addition to integrating existing work, MIRA also adds further elements where necessary.

Contents

6.1 Related Work

A broad variety of blue prints of a requirements specification has been developed, from rough outlines [ISO11b] to more detailed artifact reference structures [RR06], [PE12], [DTW12], [MFP14].

The ISO/IEC/IEEE 29148:2011 standard [ISO11b] provides example outlines for a stakeholder requirements specification, a system requirements specification and a software requirements specification. The VOLERE template [RR06, p. 226ff] provides a detailed outline of a requirements specification. Furthermore, the main contents described in the VOLERE template are related in an artifact reference structure, there called requirements knowledge model. The AMDiRE approach [MF11, MFP14] defines the main elements of an artifact reference structure and provides a domain-independent (for embedded systems and information systems) instantiation of this artifact reference structure. The ARAMiS model [PE12] is an artifact-oriented RE approach developed specifically for the cyber-physical systems domain. These approaches do not define in detail how the requirements specification is embedded into the system development. AutoRAID [Sch09] provide such an embedding, but does not specifically support the validation of requirements or a design-independent formalization (see Chapter 3). None of these approaches provides a precise, design-independent formal representation of the functional requirements. The MIRA artifact reference structure defines such a formal representation and embedding in a seamless development approach.

6.2 The MIRA Artifact Reference Structure

For the description of the structure of the MIRA artifact reference structure, UML class diagrams [OMG11] are used[1]. Abstract classes are named in italic in the diagrams and denote umbrella terms used in the thesis. A *generic class* defines the set of attributes that is common for all concepts of a content item. Every concept is a specialization of this generic class, thereby inheriting all its attributes. Additional attributes can be added to each specialized concept. This inheritance mechanism facilitates the extension of the MIRA artifact reference structure with additional concepts and attributes, as demonstrated in a case study in Section 9.2.

Keys facilitate to identify objects. *Unique keys* facilitate an unambiguous identification. A unique key $ukey$ defines a set of attributes of a class that provides a unique identification of every object of that class. If two objects have the same unique key, they are identical.

The concept of a *candidate key* is a relaxation of the unique key. It accommodates for the fact that terms used in RE may not have a single meaning as they stem from different sources. For example, an external system may have the same identifier as the organization that is developing it; two departments of a company may use the same term for two completely different domain concepts. A candidate key $ckey$ defines a subset of attributes of a class. The attribute values of that candidate key identify an object of that class non-uniquely; other objects of that class may contain

[1] The tool used is Papyrus (https://eclipse.org/papyrus/)

the same candidate identifier. The identification has to be performed externally, for example, by a person.

The main characteristics of the MIRA artifact reference structure have been presented in [TH15] and [AVT+15].

6.2.1 Content Items

The concepts of MIRA are grouped into four content items. The *system context* describes relevant elements in the context of the system under development; a *requirement list* defines the different conditions and capabilities that the system under development should meet; a *trace link list* documents the associations between concepts; a *QA collection* facilitate to document QA activities and results.

6.2.2 System Context

The system context is the operational and business environment of the system under development [PE12]. It does not belong to the system and therefore cannot be influenced during development, but surrounds the system once it has been deployed [DTW12].

Descriptive Element. A `descriptive element` defines the common information for all concepts in the context of the system under development, see Figure 6.1.

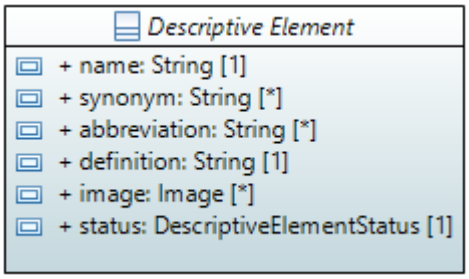

Figure 6.1: Attributes of a descriptive element

Each descriptive element has an identifying *name* along with optional *synonyms* and *abbreviations*. Each element is further described by a mandatory *definition* of this term in the problem domain and by optional (informal) *images*. A *status* defines whether the elements are *New*, *In Consolidation*, or *Consolidated* with the stakeholders of the system, see Figure 6.2.

Descriptive elements may stem from different domains, companies or departments. These different people may name different concepts identically. For example, stakeholders from different companies may have the same name, and therefore two different stakeholder objects with the same name are instantiated. A term might have various meanings, depending on the department that uses this term. Two documents

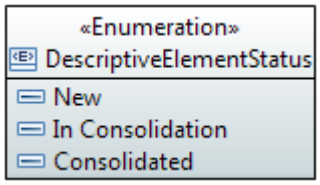

Figure 6.2: Status of a descriptive element

may have the same name in different companies. Therefore, the name of a descriptive element cannot be assumed to be unique. The same yields for synonyms and abbreviations. Candidate keys to identify a descriptive element are its name, the set of synonyms and the set of abbreviations.

A descriptive element is an abstract concept that is not instantiated. In Figure 6.3, it is therefore modeled as a generic class. This figure provides an overview of all specializations of the descriptive element. As all of these concepts are descriptive elements, they have the attributes of a descriptive element that were defined above. Some concepts have additional specific attributes. Descriptive elements are specialized to the following concepts:

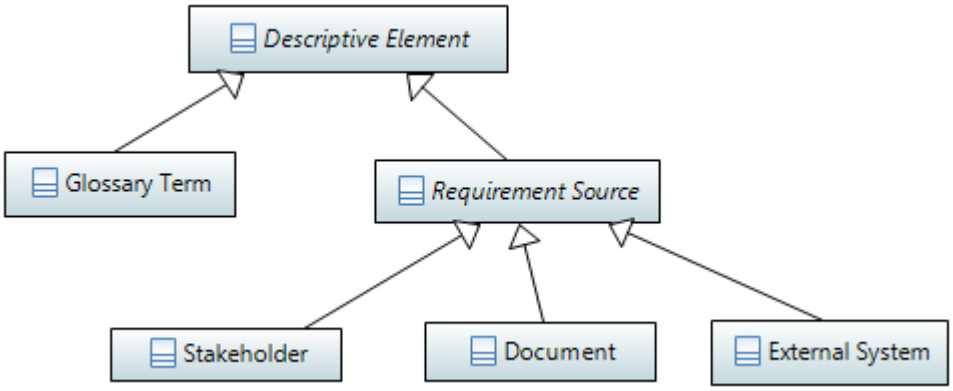

Figure 6.3: Concepts specialized from the descriptive element

Glossary Term. A `glossary term` records a domain- or project-specific term and its definitions. Glossary terms are used to capture the vocabulary of the problem domain. Glossary terms have all attributes of a descriptive element, but no further, specific attributes.

Requirement Source. A `requirement source` is the origin of a requirement [Poh10]. Following Pohl [Poh10], the requirement sources are differentiated in three types:

Stakeholder. A `stakeholder` is anyone with an interest in or an effect on the outcome of the system under development [RR06]. For each stakeholder, his/her

contact information can be documented (for example name, email, telephone number), see Figure 6.4.

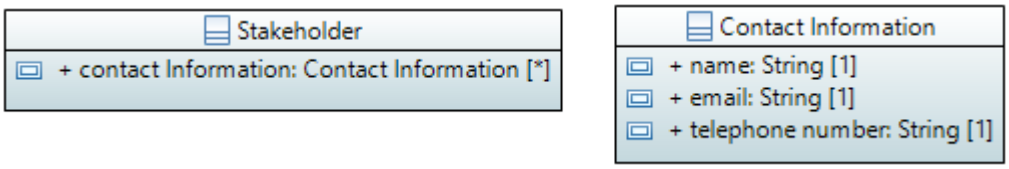

Figure 6.4: Attributes specific to stakeholders

Document. (Existing) `documents` such as standards or external system specifications are another important source of requirements. Documents include a list of *files* (the actual documents), see Figure 6.5. Each file has a short description, version information, and a path to the file.

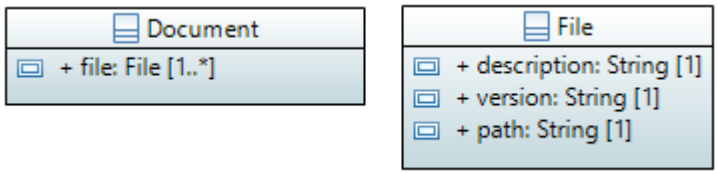

Figure 6.5: Attributes specific to a document

External System. Another important source of requirements are the `external systems` that the system under development should interact with.

In the example of a traffic light controller, stakeholders are for example the client of the traffic light controller that is the company that builds the traffic light system and therefore commissions the traffic light controller, the pedestrians and car drivers that interact with the traffic light system, but also the persons involved in the development of the traffic light controller such as software and test engineers. Documents include for example preliminary requirement collections or relevant regulatory documents and safety and security standards that apply to the system under development. External systems for the traffic light controller are the traffic lights that the traffic light controller has to control, the request buttons for the pedestrians and indicators that provide feedback to the pedestrian about their request.

6.2.3 Requirements

Requirement. A `requirement` is the generic concept for the content item 'requirement list'. In the MIRA artifact reference structure, each requirement contains typical requirement management information as proposed in Robertson and Robertson [RR06] and Kotonya and Sommerville [KS98]. The attributes of a requirement are summarized in Figure 6.6 and defined in the following.

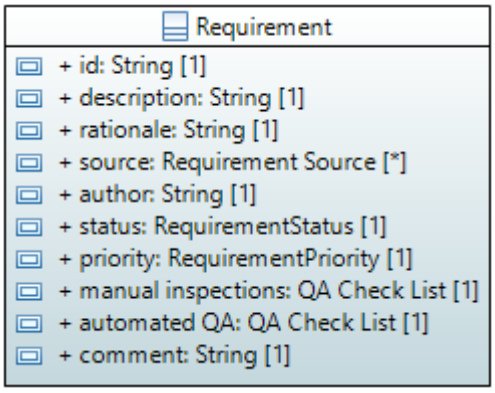

Figure 6.6: Attributes of a requirement

The *ID*, the unique key of a requirement, is needed for further references to this requirement [RR06, p. 15]. IDs are used to reference requirements, for example in trace links. The *description* is "a one sentence statement of the intention of the requirement" [RR06, p. 15]. A *rationale* gives a justification/explanation why the requirement has been included [KS98, p. 121], [RR06, p. 15]. A requirement should contain "a reference to one or more of the sources of the requirement. This helps with analysis when changes to the requirements are proposed" [KS98, p. 121]. The source of a requirement should refer to an instantiation of the concept *requirement source* as defined in the previous section. The *author* is the person that documented the requirement.

The lifecycle *status* depends on project specific processes and is therefore configurable. MIRA proposes the status values *New*, *Documented*, *Formalized*, *Analyzed*, *Validated*, *Verified* and *Rejected*, see Figure 6.7, as these status values reflect the activities of the MIRA guideline.

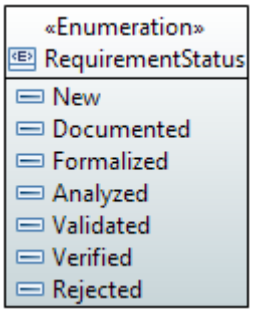

Figure 6.7: Status of a requirement

The *priority* of a requirement is "a rating of the customer value" [RR06, p. 15]. Instead of using the semantically imprecise *high*, *medium*, *low*-scale, a semantically more expressive scale like the Kano-model [Conb] may be chosen that is presented in the following. *Must-be*: The requirement is so basic that it is taken for granted when it is fulfilled; it causes dissatisfaction when the requirement is not fulfilled. *Satisfier*: The requirement results in satisfaction when fulfilled and dissatisfaction when not fulfilled. *Delighter*: The requirement results in satisfaction when fulfilled, but not in dissatisfaction when not fulfilled. Figure 6.8 summarizes the priority scale.

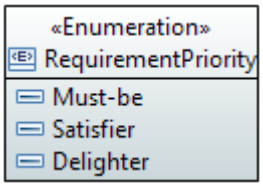

Figure 6.8: The priority of a requirement

Each requirement contains a *QA checklist* that proposes the action items that have to be conducted during the *manual inspections* of a requirement and stores the results of these action items. QA checklists are defined in detail in Section 6.2.5. Each requirement contains a *QA checklist* to store the results of *automated quality analyses*. The *comment* is an unrestricted field to document useful information that is not covered by the other fields.

Many approaches, for example, Cockburn [Coc00] or SPES [PHAB12], distinguish between solution-independent requirements, also referred to as business requirements, and solution-oriented requirements, also referred to as system requirements. Business requirements are independent of a concrete system. System requirements presume that the decision to realize a concrete system has been made. MIRA concentrates on solution-oriented `functional requirements` that define the required behavior of the system under development and the interactions with its environment. Therefore, in MIRA, requirements can be specialized to specific functional requirement types corresponding to their content (similar to [RR06, p. 15]). The functional requirement types defined in MIRA are `scenario`, `interface requirement` and `user function`. Furthermore, `goals` can be specified as they provide a rationale for the functional requirements. A requirement can remain unclassified, so that it is possible to document requirements of all other requirement types in MIRA. Figure 6.9 summarizes the requirement types of the MIRA approach.

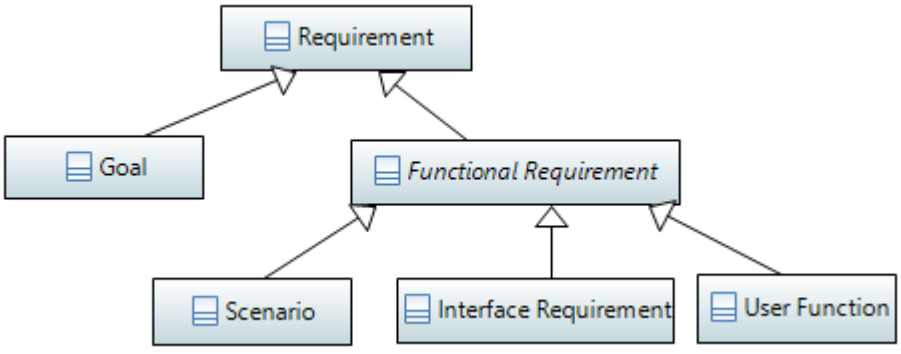

Figure 6.9: Requirement types

Each `functional requirement` can be described semi-formally by templates that provide attributes to structure the description and formally using modeling notations whose interpretation is based on the system modeling theory FOCUS, see Section 2.3.5. Goals are documented as unconstrained prose.

Goal. A `goal` captures "an objective the system under consideration should achieve" [vL01]. Goals "represent a first manifestation of the stakeholders' system vision. Goals give rationales and justifications for the functionality and features a system must possess" [PHAB12, p. 55]. As goals are prescriptive, they possess all generic attributes of `requirements`. As goals define the intent of stakeholders, they should include a reference to the intending stakeholders.

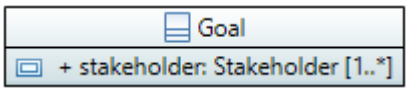

Figure 6.10: Attributes specific to a goal

For example, a goal of a pedestrian is to cross the street. A safety goal of the pedestrian (and of other stakeholders) for a traffic light system is that the traffic light system avoids collisions between pedestrians and cars by ensuring that they are never allowed to enter the crossing at the same time. Hence, the 'stakeholder' of this goal is 'pedestrian', the informal 'description' of the goal is 'The traffic light system shall prevent the collision of pedestrians and cars to ensure that pedestrians can cross the street safely'.

Scenario. A `scenario` describes a sequence of interactions of a system under development with its actors.

In MIRA, scenarios are defined as a requirement type, see Figure 6.11. Strictly speaking, a scenario is not necessarily a single requirement, but it can summarize both descriptions of the system environment and requirements on the system under development.

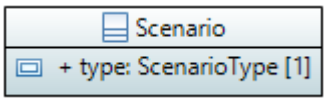

Figure 6.11: Attributes specific to a scenario

A *success scenario* describes the interactions between the actors and the system under development in case of success. Scenarios can also be used to document possible failures of the system and their handling in *failure scenarios*. Scenarios allow furthermore to document interactions that should *not* occur in *forbidden scenarios*. An overview of the scenario types is given in Figure 6.12.

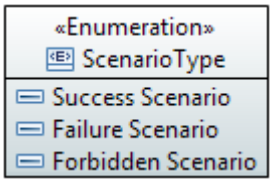

Figure 6.12: Scenario type

An example success scenario for the traffic light controller is the activation of the pedestrian light. The informal description allows for documenting the content of

the scenarios at different levels of granularity and does not constrict it further. A potential 'description' of this scenario is 'The pedestrian activates the pedestrian light of the traffic light system'. A more precise informal description is 'The pedestrian pushes a request button. The indicator of the traffic light is activated. In the next steps, the traffic light switches to yellow-red and then to red. Finally, the pedestrian light switches from 'go' to 'no-go'.

MIRA provides a semi-formal documentation of a scenario. A semi-formal scenario description consists of a course of *scenario steps*, see Figure 6.13. Each scenario step

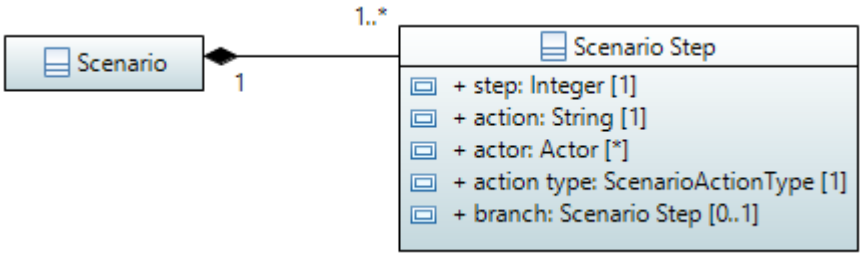

Figure 6.13: Semiformal representation of a scenario

describes an *action*. Each action is either controlled by an *actor* of the environment of the system or by the system itself and has an impact on one or more actors. Each action can have an effect either on the environment or on the system. Zave and Jackson [ZJ97] distinguish actions into shared and unshared actions. Shared actions are at the interface of the system and its environment, here input and output actions. An unshared action in the environment is not observable by the system and vice versa. Based on this classification, MIRA distinguishes four *action types*, see Figure 6.14: An *external* action is in the environment and is unobserved by the system; an *input* action describes a stimulus from the environment to the system; an *internal* action is within the system; an *output* action is a system response in the system environment. An optional *branch* of a scenario step points to a scenario step of the or another scenario. For example, a success scenario may branch to a failure scenario. This branch provides a simple means to model variations of action sequences.

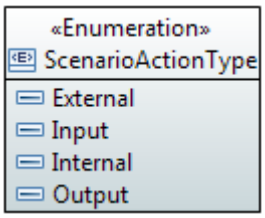

Figure 6.14: Scenario action type

The semi-formal representation of the pedestrian-light-activation scenario introduced above is provided in Table 6.1. This representation form leads to an increase in the preciseness of information by defining the action types and visualizes the lack of information about branches, for example, potential failures of the system under development.

Each scenario can be represented formally as a Message Sequence Chart (MSC). An

Table 6.1: Example for a semi-formal scenario for the traffic light controller

Step	Action	Actor	Action Type	Branch
1	The pedestrian pushes a request button.	Pedestrian	Input	
2	The indicator of the traffic light is activated.	Indicator	Output	
3	The traffic light switches to yellow-red and then to red.	Traffic light	Output	
4	The pedestrian light switches from 'go' to 'no-go'.	Pedestrian light	Output	

example for a scenario formalized as MSC is given in Figure 6.15.

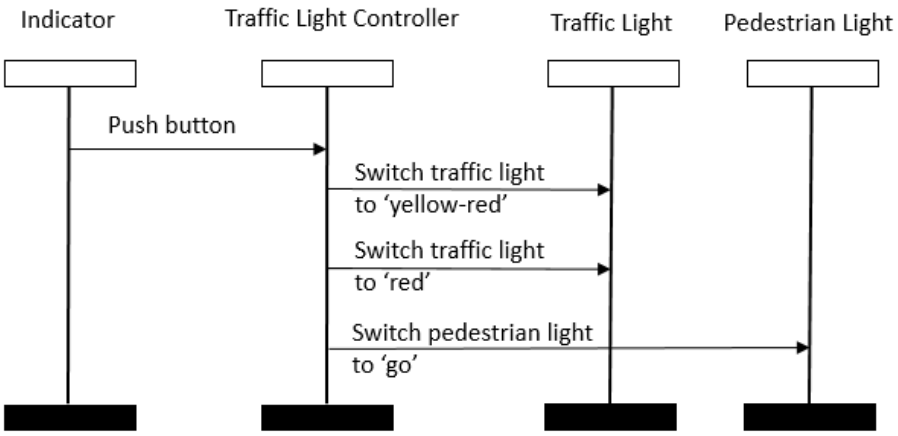

Figure 6.15: Example for a formal scenario for the traffic light controller

Interface Requirement. An `interface requirement` defines the required interface behavior of the system under development and the stimuli and reactions that are visible at the system interface.

An example for an interface behavior requirement of the traffic light controller is the requirement 'accident prevention': While the traffic light is 'green', the pedestrian light shall be 'no go'.

Figure 6.16 presents the semi-formal representation of an interface requirement. MIRA distinguishes between the syntactic interface and the interface behavior.

An interface requirement defines the syntactic interface of the system under development. This syntactic interface can be explicitly described by the list of *stimuli* and *responses* of the system under development. The stimuli include desired and undesired effects on a system from its environment. The responses comprise desired and undesired system behavior. *Data elements* capture the stimuli and reactions of a sys-

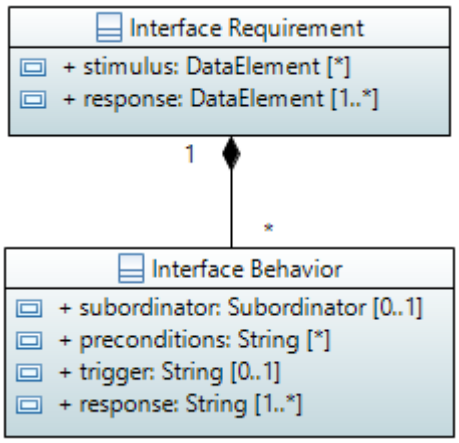

Figure 6.16: Semi-formal representation of interface requirements

tem under development. Data elements are documented by a *name* and a *value* of a *data type*. Optionally, the *actors* that cause a stimulus or should receive a system response can be documented. The *data type* is defined by a *name* and a set of *values*.

Figure 6.17: Data element

For instance, the data type 'traffic light signal' has the values 'green', 'yellow', 'red', and the 'pedestrian light signal' has values 'go' and 'no go'. Two system responses of the traffic light controller are the signals for the actor traffic light and for the actor pedestrian light, for example, 'red' for the traffic light and 'go' for the pedestrian light.

The *interface behavior* relates a set of stimuli with the required system responses. A means to structure interface behavior semi-formally are patterns for constraint natural language such as EARS [MWHN09]. An interface behavior can be ubiquitous, or its applicability can be restricted by *subordinators*, see Figure 6.18. Subordinators denote whether the requirement is event-driven (when), state-driven (while), or denoting unwanted behavior (if). These subordinators come with optional *preconditions* and *triggers*. Each interface requirement describes required *system responses*.

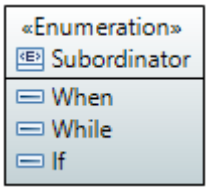

Figure 6.18: Subordinators

In the example of the interface requirement on accident prevention, the semi-formal representation distinguishes between the actual system stimulus / response and the actors. Therefore, in this representation we distinguish between the stimulus 'traffic light signal' and the actor traffic light, see Table 6.2. Furthermore, we distinguish between the system response 'pedestrian light signal' and the actor 'pedestrian light', see Table 6.3. This differentiation facilitates to express the actual requirement for the traffic light controller more precisely on the system boundaries. The interface behavior is modeled in Table 6.4. For a more precise definition of the system boundaries under investigation read also on 'design scope' in Section 6.2.4.

Table 6.2: Example for a semi-formal representation of a stimulus of an interface requirement for the traffic light controller

Stimulus	**Data Type**	**Values**	**Actor**
Traffic light signal	TrafficLightSignalValue (green, yellow, red)	green	Traffic light

Table 6.3: Example for a semi-formal representation of a system response of an interface requirement for the traffic light controller

Response	**Data Type**	**Values**	**Actor**
Pedestrian light signal	PedestrianLightSignalValue (go, no-go)	no-go	Pedestrian light

Table 6.4: Example for a semi-formal interface behavior of an interface requirement for the traffic light controller

Subordinator	**Precondition**	**Trigger**	**System Response**
While	–	The traffic light signal is green or yellow	The pedestrian light signal shall be no-go.

Interface requirements can be represented formally, see Figure 6.19. The syntactic interface can be represented as a *component* with typed *ports* that represent the stimuli and reactions. Input ports model stimuli, output ports model reactions. Formal *assumptions and guarantees* formalize value restrictions for stimuli and reactions. The interface behavior of the component can be formalized with *I/O assertions* or *state automata*.

An example for a formal representation is given in Figure 6.20. A component defines the stimuli and reactions of the traffic light system. A logical formula defines the interface behavior.

User Function. A `user function` defines a user-visible function that the system under development offers. A user function captures a set of solution-oriented requirements [VEFR12], such as scenarios and interface requirements. Each user function realizes a piece of black-box functionality and is defined by its syntactic interface and its behavioral specification.

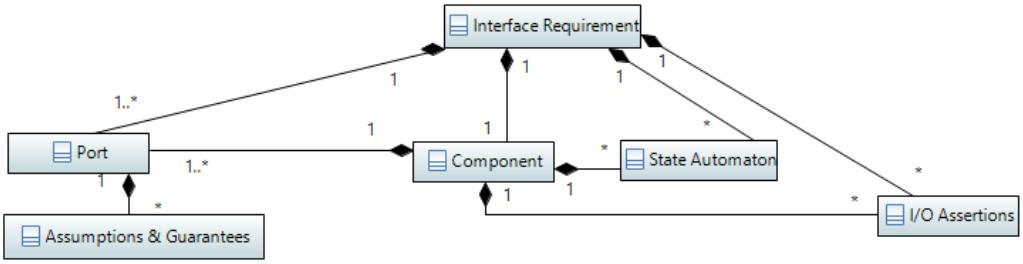

Figure 6.19: Formal representation of an interface requirement

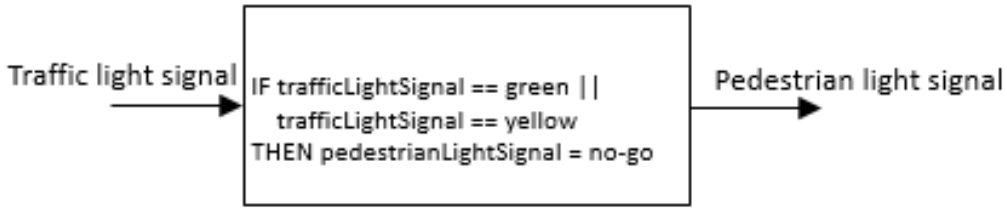

Figure 6.20: Example for a formal representation of an interface requirement for the traffic light controller

Use cases can be used to describe instances of using a system and its features in scenarios [Bro10b]. Therefore, the MIRA approach facilitates a semi-formal description of a user function by the "fully dressed use case" format presented in [Coc00]. A use case describes, how the system under development responds to one of the actors in order to achieve a particular *goal*. This includes all steps from the trigger to goal delivery, inclusive any "clean-up" [Coc00] afterwards. These steps form a *scenario*. Optionally, a use case can contain one or more scenarios. MIRA uses the same representation for scenarios in a use case as for the concept scenario that was introduced above. Figure 6.21 presents the semi-formal representation of user functions.

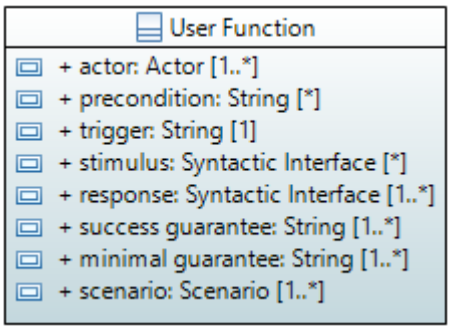

Figure 6.21: Semi-formal representation of a user function

Actors are those stakeholders and external systems that interact with the system under development in the course of using the user functions. The actors are a subgroup of the stakeholders and external systems that have been defined in the system context. Therefore, the user function should refer to them. The *precondition* states the

conditions under which a user function can be executed. The *trigger* describes how a user function is initiated. The MIRA approach facilitates to assign *scenarios* to use cases. *Stimuli* and *reactions* describe the syntactic interface for the interactions of the system under development with its environment on the system boundaries. Documenting stimuli and reactions is an extension to the "fully dressed" [Coc00] format. *Minimal guarantees* describe the "fewest premises the system makes to the stakeholders, particularly when the primary actor's goal cannot be delivered" [Coc00]. The *success guarantees* describe the conditions under which a user function terminates in case of success. Success guarantees add to the minimal guarantees.

A main user function of the traffic light system is the 'traffic light circuit function'. Use cases describe the different uses of the system that use this function. The use case 'pedestrian crossing' describes the use by a pedestrian, refining the example goals provided above. Table 6.5 presents a use case describing this user function semi-formally. The 'activate the pedestrian light' scenario introduced above is the main success scenario of this use case that leads to the success guarantee. An alternative success scenarios can describe, how the system reacts when the pedestrian light is already 'go'. Failure scenarios describe the interactions of the system with its environment in case of failures.

Table 6.5: Example for a semi-formal user function of the traffic light controller

Actors	Pedestrian, request button, pedestrian light, traffic light, indicators
Precondition	The traffic light system is in operating mode.
Trigger	The pedestrian pushes the request button.
Stimuli	Request button signal
System Response	Signals to the indicators, the traffic light and the pedestrian light
Minimal Guarantee	The traffic light remains 'green' and pedestrian light is 'no go'.
Success Guarantee	The traffic light is 'red' and pedestrian light is 'go'.

Figure 6.22 presents the formal representation of a user function. A *component* represents the user function. Typed input and output *ports* represent the stimuli and reactions of the user function. Each scenario of a user function can be modeled as an MSC. The required system behavior defined in a user function can also be modeled in a state automaton.

An example for the formal representation of a user function in a state machine is given in Figure 6.23.

6.2.4 Trace Links

Trace Link. The MIRA artifact reference structure facilitates tracing requirements to other requirements, to other RE concepts, and to results of the subsequent sys-

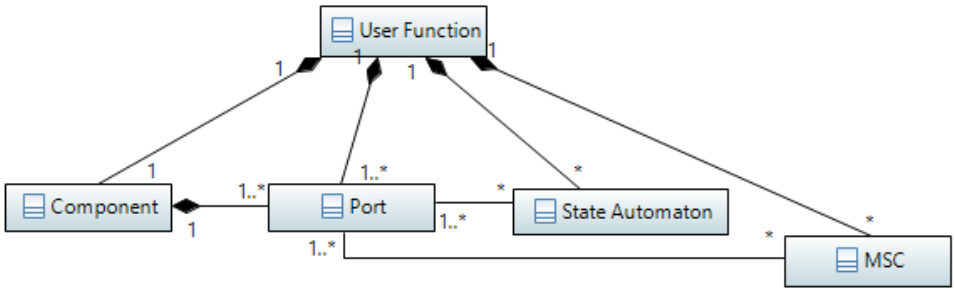

Figure 6.22: Formal representation of a user function

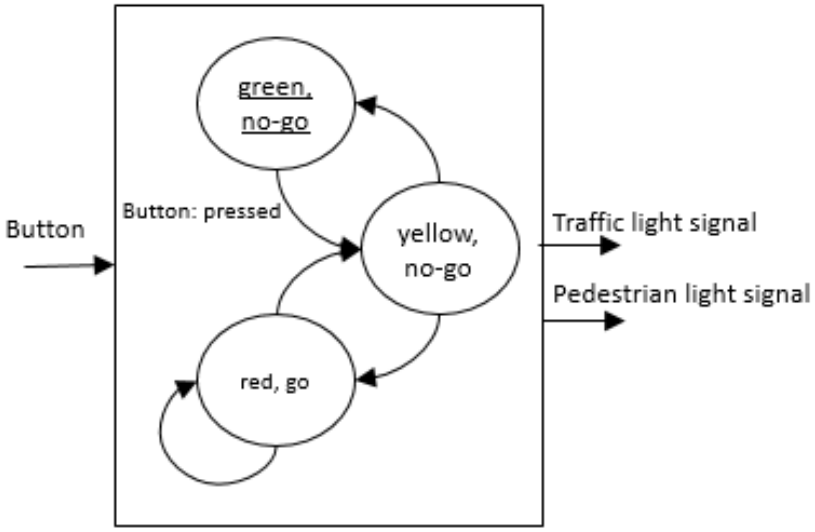

Figure 6.23: Formal representation of a user function

tems or software engineering phases that are documented in the functional, logical or technical viewpoint. In MIRA, a `trace link` documents these associations.

A trace link has the following attributes, see Figure 6.24: The *status* of a trace link

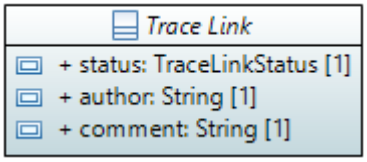

Figure 6.24: Trace link management attributes

corresponding to the traceability process. MIRA proposes a status concept (see Figure 6.25) based on a simple process: Retrieving a trace link leads to a *New* link. After the creation or a potential update, the requirements engineer decides whether the link needs validation and consolidation of the concerned stakeholders. For that purpose, the link has the status *In Consolidation*. After the consolidation, the link has the status *Consolidated*. The *author* indicates who created the link. In the actual ver-

sion of MIRA, all links are created manually. In the RE literature, complementary approaches for the semi-automatic or automatic link generation exist (see for example [GCHH+12b]), where the author may be an algorithm. The *comment* is an unrestricted field to document useful information that is not covered by the other fields.

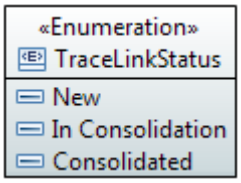

Figure 6.25: Status of a trace link

A link *source* is "the artifact from which a trace originates" [GCHH+12b]. In extension to [GCHH+12b], not only one artifact but a set of artifacts may denote the origin. A link *target* is "the artifact at the destination of a trace" [GCHH+12b]. The destination can be one or more artifacts. Sources and targets of trace links should be documented as a reference to the source and target artifacts. The unique key to identify a trace link consists of the triple type, sources and targets. No reason could be identified to distinguish between trace links where these three pieces of information are identical.

On the coarse grain level, MIRA distinguishes trace links depending on the type of their sources and targets, see Figure 6.26. When both sources and targets are requirements, trace links are called *requirement trace links*. *External trace links* link requirements to other viewpoints. The link semantics define "the purpose or a meaning

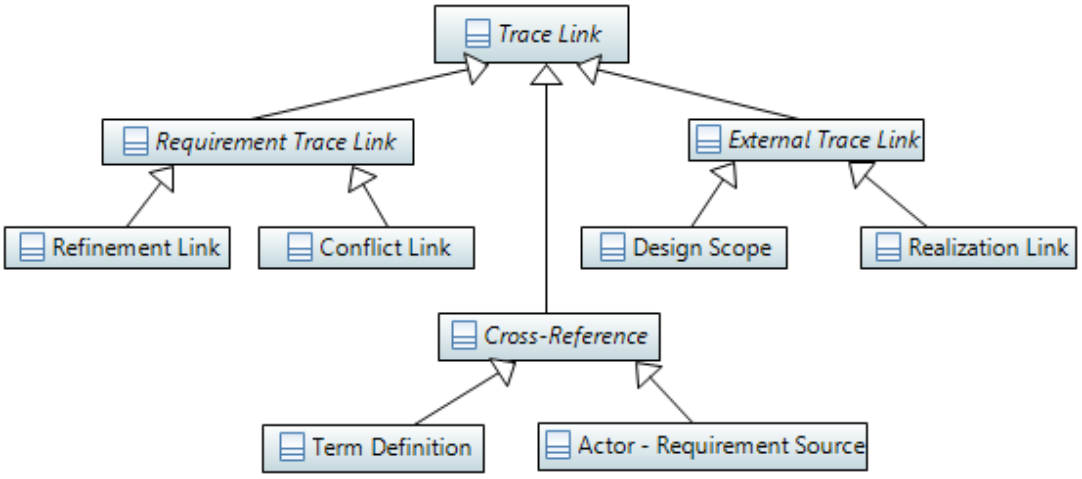

Figure 6.26: Trace links

of the trace link" [GCHH+12b] and are specified by the trace link *type*. More fine-grained, MIRA distinguishes trace links according to their meaning. Depending on the type, a link may be *directed* or *undirected*. In extension to the fundamental literature on trace links [GCHH+12b], depending on the trace link type, trace links cannot only be defined bilateral, for example, between two requirements, but also multilateral.

Requirement Trace Link. When source and target artifacts of a link are requirements, then this link is called a `requirement trace link`. Requirement trace links can be distinguished into refinement and conflict links.

Refinement. Refinement, see Figure 6.27, is the "stepwise process of adding more information" [BS01, p. 6]. `Refinement links` are always directed links from the refined requirements to the refining requirements. In MIRA, a refinement trace link can be defined between requirements of the same type and between the following requirement types: Goals are refined by functional requirements. For example, a user function should always refine the goal of a stakeholder (see Cockburn [Coc00]). Functional requirements can be refined by further functional requirements. In MIRA, a refinement link between functional requirements expresses one of the three types of refinement defined in FOCUS, behavioral refinement, interface refinement and conditional refinement (see Section 2.3.5.5). MIRA facilitates a refinement with more than one requirement as a source or a target. Nonetheless, the formal representation of a refinement is limited to one source and one target, see below.

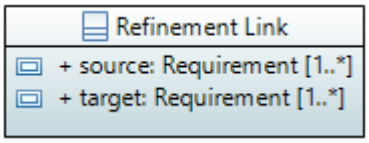

Figure 6.27: Refinement link

Conflict. A `conflict link`, see Figure 6.28, between a set of requirements is an undirected trace link that means that the linked requirements cannot be fulfilled at the same time in the system under development. For instance, conflicts with a technical background may hinder a fulfillment: A car might drive up to 200 km/h and might turn within two meters of radius but not at the same time. Another reason may be a logical inconsistency between requirements, where requirements state assertions that can never be true at the same time. A conflict can be a temporary inconsistency that has to be analyzed further in order to resolve it. For example, a conflict can be resolved by adding further preconditions to a requirement or by rejecting a requirement. If a conflict is irresolvable, the requirements are (indefinitely) contradicting each other.

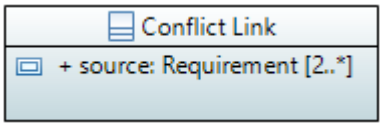

Figure 6.28: Conflict link

External Trace Link. When a requirement is linked with a model element from another viewpoint, the link is called `external trace link`. In MIRA, potential trace links between functional requirements and the functional and logical viewpoint

are defined. Further details on the dependencies to these viewpoints are provided in Section 6.3. MIRA distinguishes between design scope and realization.

Design Scope. The `design scope` of a requirement is the current system under development. The design scope determines the "size of the system under discussion" [Coc00]. The design scope, see Figure 6.29, can be defined by the name of the system under development; if the system is not yet determined, it is described by the project name. The current system under development may itself be a subsystem of a surrounding system. A system decomposition into subsystems is typically defined in the functional, logical or technical viewpoint [DTW12]. This system decomposition is the basis for developing the various subsystems in the next lower abstraction level. This system decomposition also determines the *design scope* for subsystem requirements. If the system decomposition has been explicitly modeled, the subsystem requirement can be linked to the component that represents the subsystem in the surrounding system.

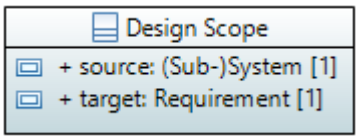

Figure 6.29: Design scope

Realization. A requirement can be realized in one or more components in the subsequent viewpoints. For example, a function in the functional viewpoint or a subsystem in the logical viewpoint realizes the behavior required by a use case or an interface requirement. The `realization link`, see Figure 6.30, connects the source, the requirement that is realized, to the target component that realizes this requirement. The link expresses that each of these components realize the requirement. A realization link indicates the need for verification: The target model element has to be verified against the requirement(s).

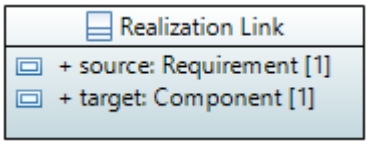

Figure 6.30: Realization

We exemplify the trace links that are introduced above using a set of requirements on accident prevention. For this example, we distinguish the two systems 'traffic light system' that includes the traffic lights and the pedestrian lights and all necessary hardware and software, and the 'traffic light controller' including software and hardware that controls the lights. The first requirement with ID 'R1' has the description 'While the traffic light is 'green', the pedestrian light shall be 'no-go". R1 has a design scope on the traffic light system. The second requirement with ID 'R2' has the description 'While the traffic light signal is 'green', the pedestrian light signal shall

be 'no-go". R2 has a design scope on the traffic light controller. R1 is a refinement of R2. Requirement R3 has the description 'While the traffic light signal is 'green', the pedestrian light signal shall be 'go". R3 is in conflict with R2, as both can never be realized at the same time. To solve this conflict, R3 is rejected. R2 is realized in the traffic light controller function 'pedestrian light function'. The various links that can be documented for this example are summarized in Table 6.6.

Table 6.6: Example for trace links for the traffic light controller

Type	**Source(s)**	**Target(s)**
Design Scope	R1	Traffic light system
Design Scope	R2	Traffic light controller
Refinement	R1	R2
Conflict	R2	R3
Realization	R2	Pedestrian light function

MIRA provides means to represent refinement links and realization links by a formal refinement specification. The precondition for formalizing a refinement link is that source and target requirements have to be functional requirements and must have a formal representation. The precondition for formalizing a realization link is that the target of a realization link, the target requirement or target component, has to be represented formally. Due to limitations of the modeling language, a refinement link can only be formalized, when it has one source. A *formal refinement specification* defines the transformations between the formal representations of source and target, the representation function that defines the mapping from the input ports of the source to the target and the interpretation function from the output ports of the target to the source. For details on the formal refinement specification see Section 2.3.5.

Cross-Reference. An association between the system context and a requirement is documented in a `cross-reference`.

Term Definition. The terms of descriptive elements, their name, abbreviation and synonym, should be referenced at every occurrence of that term in other objects. These terms are used as a part of an attribute of the type String, for example, the name of a glossary entry is used in the definition of another glossary entry. A `term definition` cross-link documents this association.

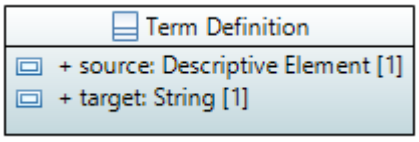

Figure 6.31: Term definition reference

Actor – Requirement Source. The actor of a scenario or user function is somebody with an interest in the system under development and therefore should be an `actor - requirement source` cross-reference to a requirement source.

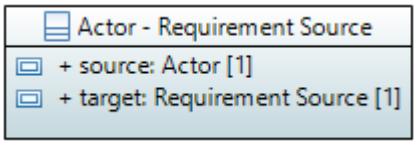

Figure 6.32: Actor - requirement source reference

6.2.5 QA Collection

The content item 'QA collection' summarizes all concepts for documenting QA activities and results. QA check lists facilitate to document activities and results for a single requirement. 'Conflict' links as introduced above document QA results that comprise more than one requirement.

QA Check List. A `QA check list` provides a means for the detailed documentation of QA activities and their results. In the current version of MIRA, QA check lists contain check items for the analysis, validation and verification of requirements. An overview is given in Figure 6.33. Every requirement has a unique check list. Depending on further QA activities, QA check lists could also be offered for other content items or groups of content items. The QA check list can be used both for manual inspections and automated checks. In the case of manual inspections, it guides the inspections. It documents the results for both manual inspections and automated checks.

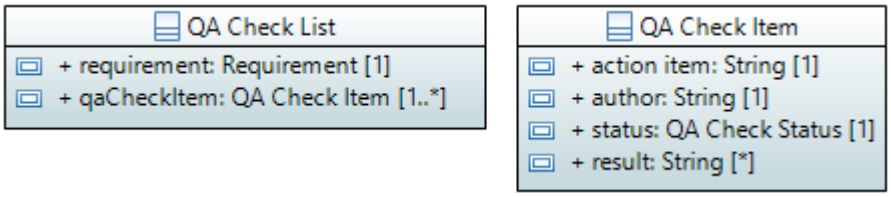

Figure 6.33: Check list with check items

A *check item* documents a check and its results. A check item consists of an *action* that defines the check that has to be performed. An *author* documents the results of these actions. The author of the results is either the person performing the check or an automated QA technique. The *QA check status*, see Figure 6.34, gives an overview whether the check described in the action has been performed (*to be checked*). If the check is automated and cannot be performed due to technical reasons, the resulting status is a *checker error*. When the check can be performed, it either *approves* the action, or the check *fails*. When the check has been performed, the *result* can be documented. Such a result could be for example a counter example that demonstrates why a check

failed. A check item is identified by the unique key consisting of the pair action item and author.

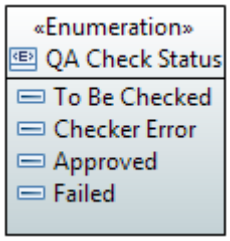

Figure 6.34: Status of a QA check list item

6.2.6 Grouping

When content items are imported from external documents, they often have a predefined grouping. It may be beneficial to retain this grouping for pragmatic usability reasons and to increase the recognition factor of the requirements specification.

Package. MIRA facilitates an optional grouping of the instantiations of all concepts in user-defined *packages* according to the content items, see Figure 6.35. The number of packages per content item is not restricted. In the traffic light example, requirements are grouped according to their design scope, thereby distinguishing requirements for the traffic light system from requirements on the traffic light controller.

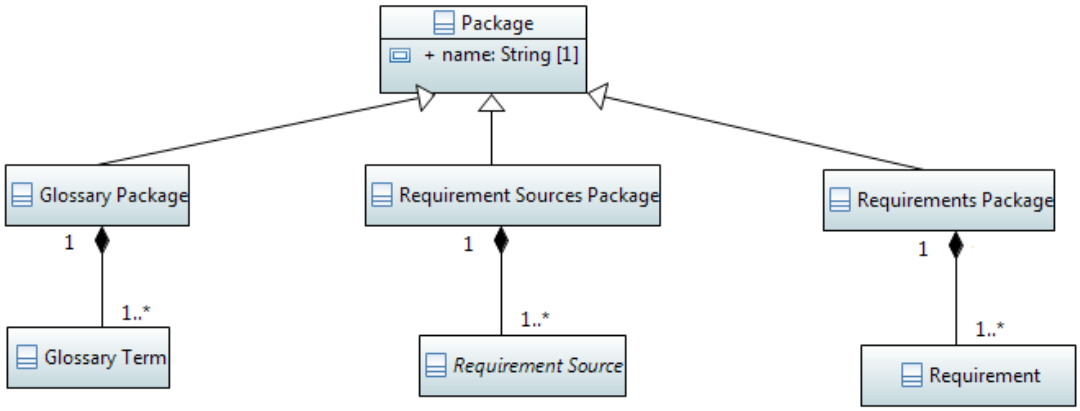

Figure 6.35: Grouping using packages

6.3 The MIRA Artifact Reference Structure in the System and Software Development

The SPES modeling framework (introduced in Section 2.3.4) defines the context of the MIRA artifact reference structure with respect to system development. The SPES

modeling framework defines the different viewpoints on a system under development and the abstraction layers of this system. SPES furthermore defines a set of artifacts for each viewpoint. MIRA provides an artifact reference structure for the requirements viewpoint. The MIRA artifact reference structure has to be embedded into the development context defined by SPES.

Following the recommendations of SPES [DTW12], at each abstraction layer the same concepts are developed in the requirements viewpoint. For requirements with a `design scope` on a subsystem, the surrounding system and its requirements become part of the 'system context'. The subsystem becomes the system under development. Other subsystems are documented as `external systems`.

The `design scope` of a `requirement` defines the concrete system under development that the requirement describes. The `design scope` corresponds to a system, a subsystem, hardware or software as documented in the logical viewpoint.

In multifunctional systems, `requirements` may be elicited based on an initial decomposition of the system into its user functions; a function hierarchy documents this decomposition in the functional viewpoint. While MIRA facilitates to document each `user function` independently in the requirements viewpoint, the functional viewpoint consolidates the user functions into a coherent system architecture of the functional behavior of the system. A `realization link` documents the realization of a `functional requirement` in a specific user function in the functional viewpoint. In the logical viewpoint, user functions are assigned to subsystems. Non-functional `requirements` can influence the system decomposition. For example, a safety requirement may lead to redundant subsystems, where each subsystem is capable to deliver the same user function. A `realization link` documents the realization of a `requirement` in a subsystem.

The dependencies between `requirements` in the requirement viewpoint and other viewpoints in SPES over the abstraction layers are depicted in Figure 6.36.

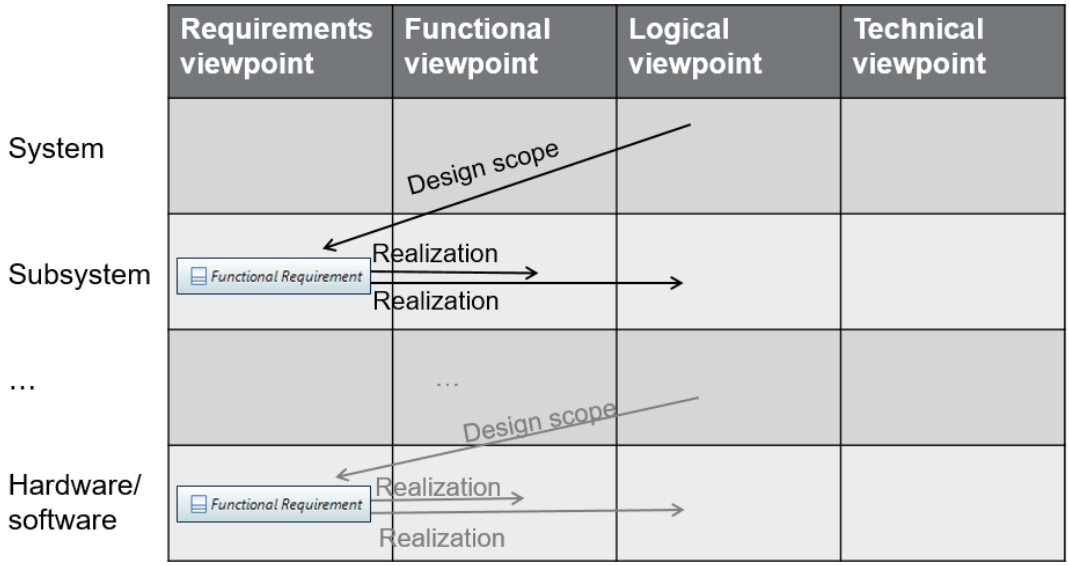

Figure 6.36: Dependencies between functional requirements and the functional and logical viewpoints

6.4 Summary and Discussion

The MIRA artifact reference structure provides a set of model elements that support the quality assurance of functional requirements. This means that each model element has a positive impact on the investigated quality assurance activities (constructive quality assurance, analysis, validation, verification). How each of these elements support quality assurance, has been investigated in Chapter 4. The model elements that are integrated in the MIRA artifact reference structure are:

1. RE content items
    - System context
    - Requirement list
    - Trace link list
    - QA collection
2. RE concepts
    - `Glossary term`, `stakeholder`, `document`, `external system`
    - `Requirement`, `Goal`, `scenario`, `interface requirement`, `user function`
    - `Refinement link`, `conflict link`, `design scope`, `realization link`, `term definition`, `actor - requirement source`
    - `QA check list`
    - Abstract concepts that cannot be instantiated and denote umbrella terms
3. Attributes
    - Attributes to manage concepts
    - Attributes to define the contents of the concepts
4. Representation forms
    - Informal, semi-formal and formal representation of functional requirements, refinement and realization links
    - Informal and semi-formal representation of other concepts
    - Semi-formal representation by attributes
    - Formal representation in FOCUS

In MIRA, trace links are defined as first class entities in the MIRA artifact reference structure; trace links are seen as an original RE concept. This has the advantage that trace links have defined attributes and that they may be formalized, if necessary.

The MIRA artifact reference structure facilitates to specify functional requirements as unconstraint prose, semi-formally and formally. Furthermore, the MIRA artifact reference structure is integrated in a seamless model-based development framework provided by the FOCUS modeling theory that is underlying the semi-formal and formal representation of functional requirements and is embedded in the SPES modeling framework.

6.4.1 Granularity

The granularity of the RE concepts defined in the MIRA artifact reference structure determines the level of detail in which the documented information is defined. For example, the artifact reference structure could define a stakeholder, but it also could define the various stakeholder groups in more details; each requirement may be traced independently or traces may link groups of requirements. Generally, we assume that a more coarse-grained artifact reference structure reduces the effort for specification and the effort for training of the persons that apply the artifact reference structure. A more detailed model may yield additional benefits with respect to quality such as more efficient analysis methods.

To give an example for more efficient analysis methods, we compared the efficiency for impact analysis of artifact reference structures with different levels of granularity in the classification of RE concepts [TH15]. An RE artifact reference structure that precisely defines trace links, source and target artifacts can impact the efficiency of the *analysis of changes* of requirements on the requirements specification. We applied an RE artifact reference structure to exclude those requirement types from a change impact analysis that cannot be impacted by a change. This can reduce the number of requirements that need to be investigated in the analysis. Thereby the artifact reference structure makes the analysis more efficient in comparison to an unclassified list of requirements where all requirements have to be investigated. Furthermore, we demonstrated that a more detailed artifact reference structure can further increase the efficiency of the analysis.

Ingram and Riddle [IR12] investigated the granularity of trace links and its influence on costs and benefits of traceability. They state that varying the granularity of trace links can influence trace costs and the quality of traceability. Determine an ideal granularity of trace links is out of scope of MIRA, as the authors assumed that determining costs and benefits of trace links depends on the project context, for example on the expected maintenance period.

6.4.2 Requirements on the System Boundary

In RE literature, it is often assumed as a best practice that requirements are documented on the system boundary [ZJ97], often also referred to as 'black-box' requirements. Nonetheless, sometimes a system under development has to be decomposed functionally already on the requirements level into modes of operation. This decomposition may stem from domain-specific standards that prescribe system modes or solution-independent characteristics of the domain.

For example, some requirements for a control software such as the traffic light controller may only be applicable if the surrounding system is not only provided with electricity, but if the control software has previously performed a 'start up' of the system. This start up may not be measurable by a sensor from outside the control software, but influences the perceptible black-box behavior of a system.

6.4.3 Standards on Trace Links

Some domains have standards that constrain the definition of a trace link and prescribe a specific workflow on trace links. For example, in avionics the DO 178C standard [DO112] defines trace links in the context of software requirements: Trace links either indicate a refinement of requirements or they associate requirements with artifacts in the further software development process (test and implementation). In cases where such standards apply, trace links that are prescribed by the standard have to be clearly distinguished from other links that are not prescribed by the standard. When applying MIRA in such an environment, the naming and definition of all `trace link` concepts have to be tailored to the definitions in the standard.

6.4.4 Validity

A model is *valid* (see [Bro13a]) with respect to a system, if the questions answered based on the model provide the same answers as asking the questions on the system. Investigations concerning an invalid model, especially the identification, correction and consequences of invalid underlying models, is not addressed in this work. The MIRA artifact reference structure is valid, if it effectively and efficiently supports the quality assurance of functional requirements. The individual model elements of MIRA and their impact on quality assurance were investigated in Chapter 4. The MIRA guideline in Chapter 7 provides rules to instantiate the MIRA artifact reference structure in a conformant requirements specification. The validation reflects whether the MIRA artifact reference structure is sufficient to support quality assurance, measured through a predefined set of workflows that tackle quality assurance. To implement the MIRA artifact reference structure in a tool, it has to be adopted to a Data Base Management System (DBMS), for example, a document-oriented DBMS or a relational DBMS. The implementation of the MIRA artifact reference structure in an open source tool shows the feasibility of the MIRA artifact reference structure, see Chapter 8. The case studies in Chapter 9 evaluate the effective application of the MIRA artifact reference structure on industrial specifications.

Chapter 7

The MIRA Guideline

MIRA provides a guideline for the creation and the quality assurance of a requirements specification. This guideline describes how to apply the MIRA artifact reference structure that was introduced in Chapter 6.

The research question addressed in this chapter is:

> ***RQ4:*** *How to apply the MIRA artifact reference structure for the specification and quality assurance of functional requirements?*

The contribution of this chapter is the integration of existing guidelines for the specification and quality assurance of functional requirements and a tailoring of these guidelines to the MIRA artifact reference structure. The resulting MIRA guideline is not an exhaustive 'how-to' guide. Instead, MIRA describes step-by-step the actions to instantiate the MIRA artifact reference structure in a requirements specification. Furthermore, it provides guidance on the model-based quality assurance making use of the MIRA artifact reference structure. Hence, the research question asks how to *apply the MIRA artifact reference structure* rather than how to *perform* these activities.

Contents

7.1 Integrated Approaches

The MIRA guideline integrates three guidelines for model-based RE from AutoRAID, the approach of Cimatti et al. and FOCUS. Each of the guidelines has a different emphasis. Further details on these approaches are given in Chapter 2. In the following, we describe how they are integrated in MIRA.

AutoRAID proposes a classification of requirements with respect to predefined concepts, a formalization of informal requirements to attributed requirements and analyses with automated conformance checks [Sch09, p. 21ff]. MIRA adopts these actions, the classification of requirements in Section 7.3.2, the formalization to attributed requirements in Section 7.4.1 and the conformance checks in Section 7.5.2.1.

Cimatti et al. [CRST09, CRST13] propose a classification of fragments of the requirements specification according to pre-defined concepts. For each of these concepts, Cimatti et al. propose a distinct modeling notation to use for the formalization. MIRA adopts the classification in Section 7.3.2 and the proposition of modeling notations in Section 7.4.2. Cimatti et al. present a set of QA techniques and leaves the selection of requirements and techniques to the quality assurance expert depending on concrete needs. MIRA adopts this procedure, see Section 7.5.

The formal modeling theory FOCUS provides the basis for the formal specification of functional requirements, see Section 7.4. Furthermore, FOCUS facilitates the interpretation of the formalized functional requirements regarding a system model. Thereby, FOCUS enables formal quality assurance techniques that are applied in Section 7.5. FOCUS proposes to elaborate a hierarchy of user functions as part of the requirements specification [Bro10b]. We see this hierarchy as part of the functional viewpoint (see Chapter 2) and therefore it is not part of the requirements viewpoint. Nonetheless, as an input to develop the function hierarchy, MIRA facilitates to document functional requirements including user functions. These functional requirements serve to ensure that all functional aspects are correctly implemented in the function hierarchy. The MIRA guideline provides actions to verify the function hierarchy against the functional requirements in Section 7.5.4.

7.2 Overview

The MIRA guideline gives instructions for the documentation of the requirements specification as prose in Section 7.3, a formalization of requirements in Section 7.4 and the application of the quality assurance techniques in Section 7.5.

The MIRA guideline defines a set of actions for each activity. Each action is described by steps that lead to the desired outcome of the action. Preconditions define the mandatory sequence of the actions. The sequence of actions does not oppose a process. For instance, when following a waterfall process, an action could be completed before conducting the next action. When following an incremental or agile process, the sequence of action can be repeated for each increment or sprint with a small set of requirements.

The activities presented in this guideline can be conducted by different roles. We assume the following simple roles: The requirements engineer manages and controls the overall RE process. Either the stakeholders or, as a stand-in, the requirements engineer documents the requirements specification. To simplify the guideline, we assume that the requirements engineer documents them. The architect uses the requirements specification as an input to further develop the system under development. Many approaches such as [CRST09] introduce the roles of a domain experts and an expert for quality assurance techniques. The quality assurance expert is re-

sponsible for selecting appropriate quality assurance techniques for each quality assurance activity and conducts all techniques, where no specific domain-knowledge is required. The quality assurance expert calls in a domain expert whenever necessary. The domain expert typically validates the requirements and corrects issues that have been identified in the analysis and verification of requirements. Due to their professional background, domain experts are often not familiar with formal languages. This limitation has to be considered by the quality assurance expert.

Early versions of parts of this guideline have been published in [TBP14], [BJV+14], and [RTVH15].

7.3 Textual Specification

In the textual specification, the requirements engineer elaborates the requirements specification as prose (Section 7.3.1). As part of the documentation, the requirements engineer classifies each piece of information as one of the RE concepts defined in the MIRA artifact reference structure (Section 7.3.2). The requirements engineer documents the `requirement sources` (Section 7.3.3) and the `requirements` (Section 7.3.4). The requirements engineer refines the requirements (Section 7.3.5) and documents the `refinement links` that result from the refinement (Section 7.3.6). The requirements engineer defines the domain-specific terms used in the `requirements` by designating `glossary terms` (Section 7.3.7). A requirements engineer or an architect document `realization links` that associate `requirements` with the artifacts that realize them (Section 7.3.8). Figure 7.1 gives an overview of these actions.

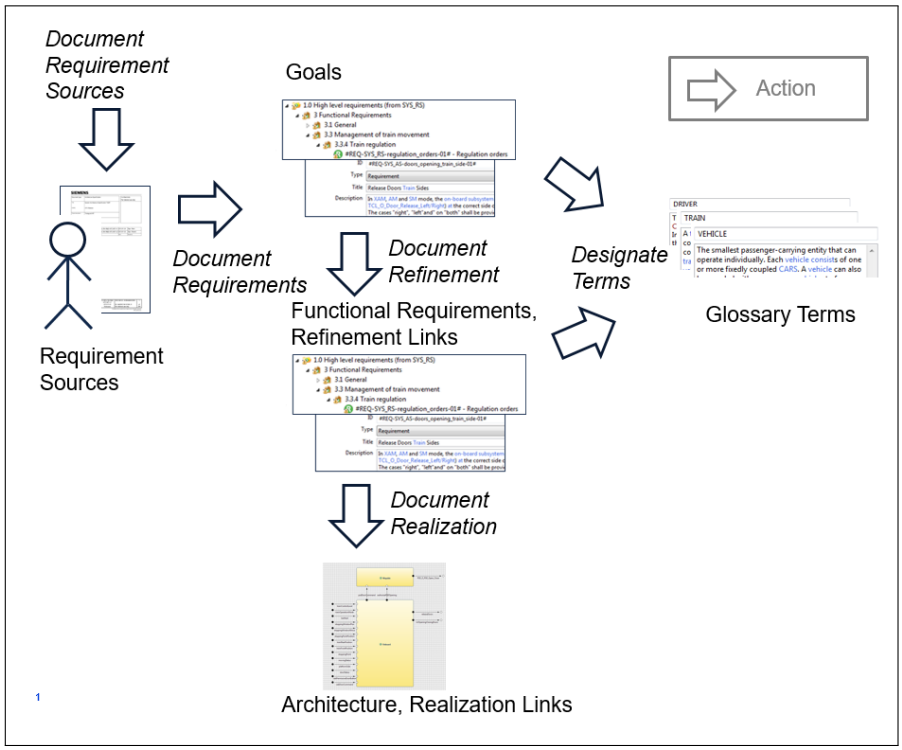

Figure 7.1: Textual Specification

7.3.1 Elaborate the Requirements Specification in Prose

The requirements engineer elaborates a coherent set of requirements. The requirements engineer documents the requirement sources and their relation to the requirements. Refined requirements are traced by refinement links. The requirements are documented using prose. For each requirement, the requirements management attributes are documented.

Preconditions. As a starting point, the requirements engineer has to identify the problem statement and some initial stakeholders of the system under development. There may also exist an undocumented candidate list of requirement sources and some draft requirements.

Outcome. The requirements specification contains a set of `requirements`, `requirement sources` and `refinement links`.

Steps.

1. Elicit the `requirement sources` and document them, see Section 7.3.3.
2. Elicit the `requirements` from the requirement sources, classify them, see Section 7.3.2, and document them, see Section 7.3.4.
   - Concrete techniques, such as interviews, to elicit `requirements` from the `stakeholders` can be found, for example, in [vL09, p. 76ff].
   - `Documents` may contain `requirements` that have to be incorporated in the requirements specification. An example is a system requirements specification as an input for a software requirements specification. If the requirement source is a document, classify the contents of this document, see Section 7.3.2, and document them according to the classification.
   - `Requirements` pertaining from `external systems` are often elicited from the responsible stakeholders or documents.
3. Refine `requirements` (Section 7.3.5) and document the `refinement links` (Section 7.3.6).
4. The *actors* of the `scenarios` and `user functions` may impose further requirements. Actors therefore have to be added as a (potential) `requirement source` to the list of requirement sources. An *actor* can be a `stakeholder` or an `external system`.
5. Document the refinement of requirements as a `refinement link`, see Section 7.3.6.

7.3.2 Classify the Elements of a Document

The requirements engineer breaks-down and assigns the contents of a document to the RE concepts defined by the MIRA artifact reference structure.

Preconditions. Unclassified contents.

Outcome. Contents are classified and documented.

Steps.

1. Break down the unclassified contents into individual objects until they match to the definitions for RE concepts provided in the MIRA artifact reference structure.
2. If an object matches a definition, document it with the management attributes that are defined for this RE concept in the MIRA artifact reference structure. If the object is a `requirement`, it has to be further classified as a concrete requirement type if possible.
3. If a stakeholder has a concrete requirement on the behavior of a system that matches a definition of one of the functional requirements, classify it accordingly as a `scenario`, `interface requirement`, or `user function`.

The requirements engineer might encounter contents that have to be documented, for example, because they are required in a development activity, and that cannot be matched distinctively to a definition of an RE concept even after breaking them down. Either none or too many definitions may be suitable. This indicates that the requirements engineer has to adapt the MIRA artifact reference structure to the project context.

7.3.3 Document a Requirement Source

Preconditions. The requirements engineer identified a requirement source, but has not yet documented it.

Outcome. The `requirement source` is documented.

Steps.

1. Classify the `requirement source` as a `stakeholder`, `external system` or `document`.
2. Create a `stakeholder`, `external system` or `document` object and assign a *name* to it.
3. Set the *status* to *New*.
4. Document the *definition* of the requirement source.
5. If the *name* of the `requirement source` has *synonyms* or *abbreviations*, document them.
6. Fill in additional information, if required: For `stakeholders`, add the *contact information*. For `documents`, add *file information*.

7. When the information of a `requirement source` is documented, set the *status* of the requirement source to *In Consolidation*.

7.3.4 Document a Requirement

Preconditions. The requirements engineer identified a requirement, but has not yet documented it.

Outcome. The `requirement` is documented as prose, including the management information and trace links to its source and to its design scope.

Steps.

1. Create a `requirement` object and assign an *ID* to the requirement.
2. Document the description of the requirement as prose in the *description* field.
3. Add a *rationale* of the requirement, the *author* and the *priority* of the requirement.
4. Classify each requirement as a scenario, an interface requirement, a user function, if the requirement matches the definition of these requirement types. If it does not match, classify it as a requirement.
5. The `requirement source` object is referenced in the *source* field of the requirement. If the requirement source is not documented yet, a new `requirement source` object has to be created.
6. Document the `design scope` for each `requirement`:
   - Create a `design scope` link. Document the system name to which the requirement refers as the *source* in the design scope link. Document the requirement as the *target*.
   - Define the design scope as a `glossary term` in the glossary.
7. Set the status of the requirement to *New*.

7.3.5 Refine Requirements

`Goals` can be further refined to a set of `functional requirements`. These can be manually synthesized into `user functions`.

Preconditions. A set of `goals`, `interface requirements` or `scenarios` are defined.

Outcome. Parts of the required behavior of the system are consolidated in a `user function`. If the complete behavior of that functional aspect is given, the `user function` can be formalized as an executable formal model, for example, a state automaton.

Steps.

1. Refine `goals` to `scenarios` and `interface requirements`. This refinement requires decisions on the system scope including system boundaries. Use `scenarios` to define, how the system under development interacts with actors. Use `interface requirements` to define the black-box system behavior necessary to fulfill these scenarios.
2. If necessary, refine the required system behavior, the system interfaces or by introducing additional input assumptions (see Section 2.3.5.5). Document the refinement in additional `scenarios` and `interface requirements` and document the `refinement link`.
3. `Interface requirements` and `scenarios` with the same level of detail of their interfaces can be synthesized as a `user function`. The `user function` provides the user-visible function that is necessary to fulfill these requirements. The user function must fulfill all dedicated `interface requirements` and `scenarios`. While `interface requirements` and `scenarios` typically only describe a partial system behavior with respect to system inputs and outputs, a `user function` can be used to describe the total behavior of a this set of inputs and outputs. In MIRA, a decomposition of a system into user functions is subject to the decisions of the functional architect, see Section 6.3. Hence, also the synthesis of `interface requirements` and `scenarios` to user functions is subject to these decisions and therefore is performed manually.
4. Document the synthesis from `interface requirements` and `scenarios` to `user functions` as a `refinement link`, see Section 7.3.6.

The MIRA approach does not explicitly define the targeted level of detail of the refinements.

7.3.6 Document a Refinement Link

Preconditions. The requirements engineer documented `requirements` and refined them.

Outcome. The `refinement link` is documented.

Steps.

1. Create a `refinement link` object.
2. Document the refined requirements as the *source* and the refining requirements as the *target*.
3. Document the *author* of the `refinement link`.

7.3.7 Designate Terms Used in the Requirements

"The only way to establish the meaning of a primitive term is to provide an informal explanation of it" [ZJ97]. Zave and Jackson [ZJ97] call documenting the meaning of a term 'designation'. They argue that a designation grounds formal representations of a requirement in the real world.

Preconditions. Requirements are specified as prose.

Outcome. The meaning of all domain-specific terms that are used in the requirements is documented as `glossary terms` or `requirement sources`.

Steps. Identify all domain-specific terms used in the description and the attributes of a `requirement`. Domain-specific terms comprise at least the following: *Actors*, the *design scope* and all system stimuli and responses in the *description* of the requirement. All domain-specific terms in the semi-formal and formal representation of a requirement should be added. Domain-specific terms can be `requirements sources`, or are documented as `glossary terms`. For example, an actor constitutes a potential `requirement source`.

1. Classify the term as a `requirement source` or a `glossary term`.
2. If the term constitutes a `requirement source`, see Section 7.3.3.
3. If the term constitutes a glossary term, create a `glossary term` object.
4. Describe the `glossary term` by a clear and precise *description*.
5. If required, add *abbreviations* and *synonyms*.
6. Set the *status* of the `glossary term` to *In Consolidation*.

7.3.8 Document a Realization Link

This activity documents the realization of functional requirements in subsequent development artifacts.

Preconditions. Requirements are specified textually, classified and realized in a subsequent development artifact.

Outcome. For each `functional requirement`, the architect documents `realization links` that indicate which subsequent development artifact realizes the requirement.

Steps.

1. Document the realization of `functional requirements` as a `realization link`. Document the requirements as the *source* and the

development artifact or a part of it as the *target*. Document the *author* of the link.

7.4 Formal Specification

The requirements engineer formalizes the `functional requirements`, their `refinement links` and `realization links` that were documented in the textual specification Section 7.3 according to the MIRA artifact reference structure.

In the first step, the requirements engineer augments each `functional requirement` by a set of attributes that structure and detail the requirement's *description* (Section 7.4.1). This intermediate step facilitates documenting the information required for the formalization independently from any formal modeling language. Hence, this information can be understood and validated by all stakeholders.

In the next step, the requirements engineer formalizes the attributed `functional requirements` (Section 7.4.2). Rules define how to transform the attributes into elements of the formal specification. The `functional requirements` are formalized independently from each other. Every `functional requirement` is represented by one or more formal specification.

In a non-trivial project, not all requirements can be represented as a formal specification. This is not possible for example, if the modeling language or its tool support are not suitable for a certain requirement type (in the case of the current AutoFOCUS3 version for example many types of extra-functional requirements) or if quality issues cannot be resolved. Therefore, textual requirements need to co-exist with formal requirements.

In order to verify a formal `functional requirement` or a formal development artifact against a formal `functional requirement`, the `functional requirement` has to be mapped to the development artifacts. A formal refinement specification provides this mapping for `refinement links` and `realization links` (Section 7.4.3). Figure 7.2 gives an overview of these actions.

7.4.1 Structure Functional Requirements

Structure and detail the informal description of a `functional requirement` by attributes and documented this structured information.

Preconditions. `Functional requirements` are described by prose (Section 7.3.2).

Outcome. `Functional requirements` are documented as structured text as defined in the MIRA artifact reference structure.

Steps. The informal description of a `functional requirement` is manually analyzed in order to extract the information that should be documented in the type-

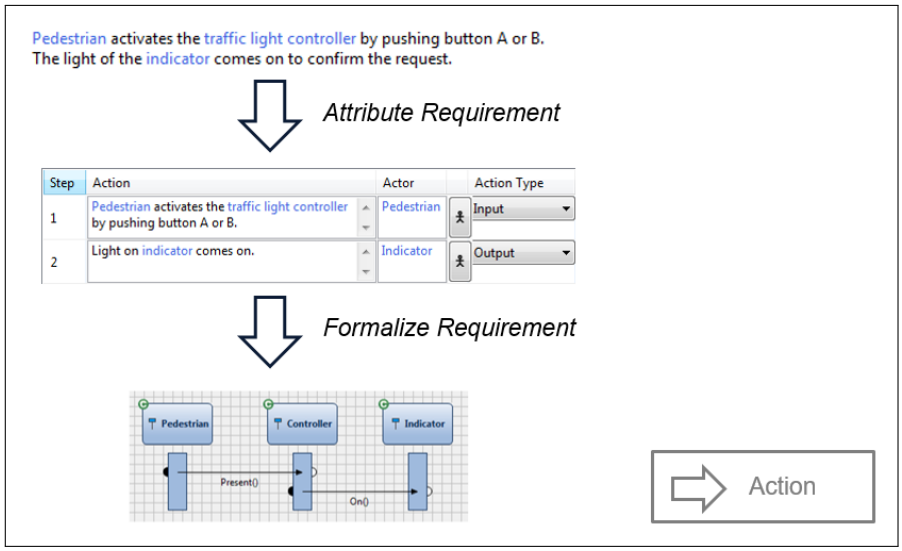

Figure 7.2: Formalization of a `functional requirement`: In an intermediate step, natural language requirements are structured and detailed by attributes using a template. The attributes can then be transformed to the formal representation of the requirement

specific attributes of the `functional requirement`. If a piece of information is not contained in the informal description, this piece of information has to be elicited and documented. For example, the piece of information can be elicited from the refined `requirements` or the *source* of the `requirement`. New domain-specific terms have to be designated (Section 7.3.7). An example for the step of an informal description to a structured `requirement` is given in Figure 7.3.

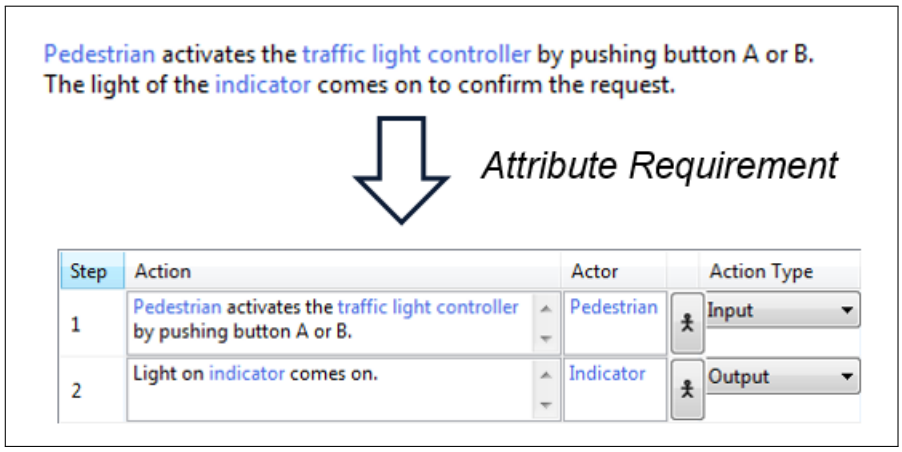

Figure 7.3: A `scenario` is structured by filling a template. The template guides the formalization by extracting and explicitly documenting the information contained in the description

7.4.2 Formalize Functional Requirements

Represent `functional requirements` as a formal specification using an AutoFOCUS3 modeling notation. Please refer to Chapter 2 for details of the formal modeling notations and their elements.

Preconditions. The information of a `functional requirement` has been structured (Section 7.4.1).

Outcome. Parts of the information documented in a `scenario`, `interface requirement`, or `user function` are represented formally. As the result of a successful formalization, quality issues of a `functional requirement` that hinder a formalization are resolved. These quality issues include ambiguous textual requirements and logical incompleteness.

Steps.

1. In the first step, the requirements engineer formalizes the `functional requirement` depending on its type. For each requirement type, transformation rules guide the transformation from the structured requirement to its formal representation.

   A `scenario` is formalized by a Message Sequence Chart (MSC):

   a) For each *actor* of the `scenario`, an MSC entity is created.

   b) For each *action* of the `scenario`, where the *action type* is an *input* or *output*, a message with entry and exit points to the MSC entities is created.

   c) In preparation of the MSC compability check between MSC and a component model, the entry and exit points of each MSC entity have to be assigned to the input and output ports to the corresponding components in the component model.

   Fig. 7.4 gives an example for a formalization of a `scenario` to an MSC.

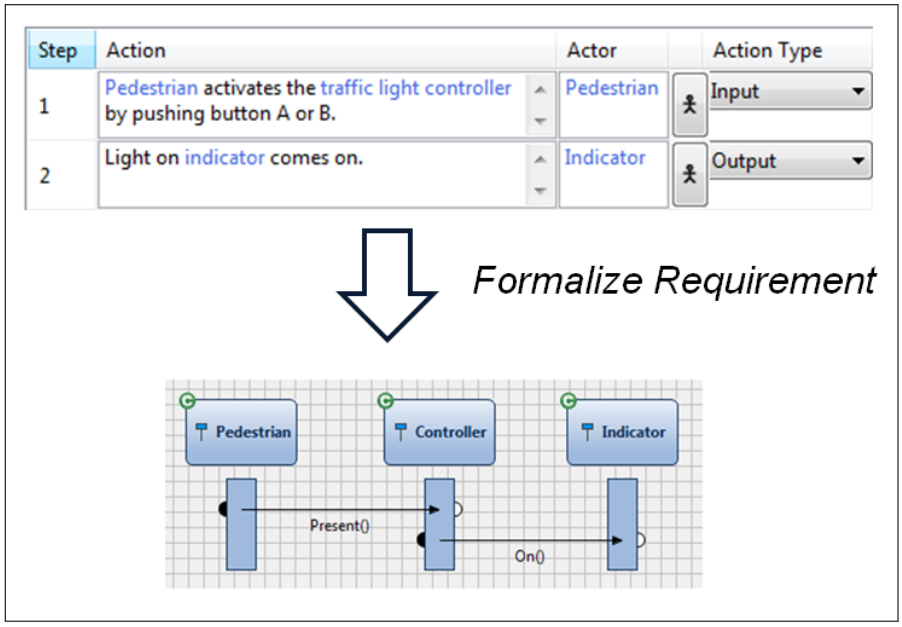

Figure 7.4: The `scenario` is formalized by mapping rules based on the template. For example, the entries in the column 'actor' are transformed to the entities of the MSC

For each `interface requirement`, its *syntactic interface* is modeled by a component diagram. The required *interface behavior* is modeled as I/O assertions.

a) For each *stimulus* and *reaction* defined in the *syntactic interface* of the `interface requirement`, create or select a data type. If a *stimulus* or *reaction* A is expected, the data type is documented with the values A and $not A$. All data types are collected in a data dictionary.

b) Formalize the *syntactic interface* of the `interface requirement` by modeling each *stimulus* as an input port, each *reaction* as an output port of a component diagram and associate the ports with the corresponding data types.

c) Model the required *interface behavior* as an I/O assertion.

`User functions` that are described by a *use case* with *scenarios* can be formalized by MSCs. Alternatively, a `user function` can be formalized as a state automaton.

2. The quality assurance expert has to be manually verified that the formal representation of a `functional requirement` corresponds to the representation as structured text.
3. The requirements engineer replicates information during the formalization. Depending on domain- and project-specific settings, the requirements engineer must eliminate this redundancy in some settings by removing the corresponding *description* and the structured text specification. Redundant information in the different representation forms of a requirement is discussed in Section 7.6.
4. The *status* of the `requirement` is set to Formalized.

7.4.3 Specify Formal Refinements

In Section 7.3, the requirements engineer documented `refinement links` between `functional requirements` and in Section 7.3.8, he requirements engineer documented `realization links` from `functional requirements` to subsequent design artifacts. The requirements engineer formalizes these `refinement links` and `realization links`. This constructively identifies quality issues in the refinements and enables formal techniques such as refinement testing [BMR12]. For more information on formal refinement specifications see Chapter 2.

Preconditions. A `refinement link` between `functional requirements` or a `realization link` between a `functional requirement` and a subsequent development artifact is documented. *Source* and *target* of these links are both specified formally and have both a design scope on the same system, subsystem, hardware or software.

Outcome. The mapping rules between *source* and *target* are specified formally. The representation function defines the transformation of the *stimuli* of the *source* to the *stimuli* of the *target*. The interpretation function defines the transformation from the

responses of the *target* to the *responses* of the *source*. Quality issues occurring during the formalization are resolved. Figure 7.5 illustrates how a formal refinement specification maps a `requirement` and its realization, a logical model of the system under development, by a representation function and an interpretation function.

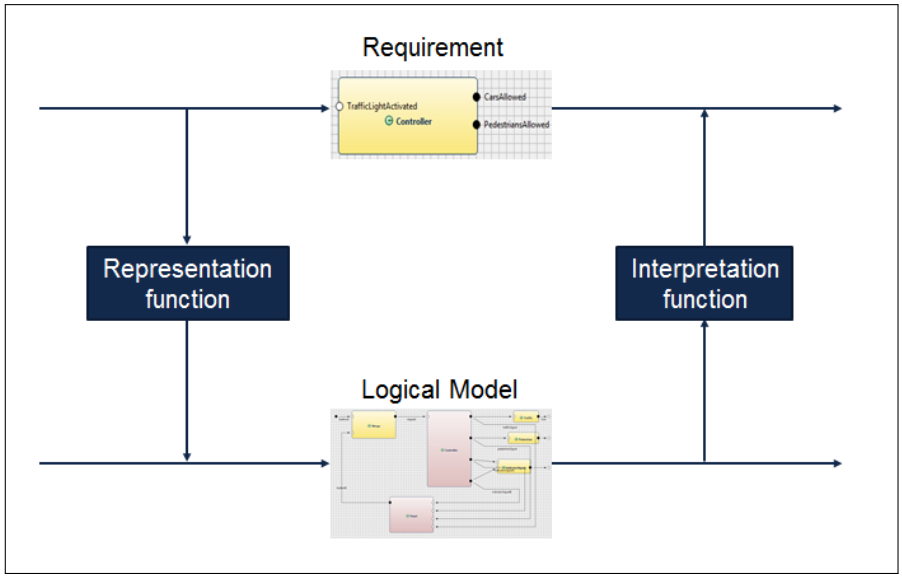

Figure 7.5: Schema of a formal refinement specification

Steps.

1. Create a formal refinement specification for the `refinement link` or the `realization link`.
2. Define the representation and interpretation functions.

If a *stimulus* or *response* of a *source* cannot be matched to its counterpart in the *target*, it has to be checked whether the `refinement link` / `realization link` is correct. For the formal refinement specification, the following conditions should hold: Input and output ports defined in the formal representation of the *source* or *target* should be used in the functions of the formal refinement specification. An uncovered port of the *source* indicates a missing interface refinement of the system under development and subsequently missing interface behavior. An uncovered port of the *target* indicates missing information in the source requirement.

7.5 Quality Assurance

Depending on the representation of a `functional requirement` as prose, structured text or as a formal specification, and with corresponding tool support, different quality assurance techniques are enabled. An informal specification can only be manually reviewed. Structured text or a formal representation facilitates model-based techniques. Structured text or a formal representation enable to perform automated database queries. A formal representation is the precondition for formal techniques. Figure 7.6 gives an overview of these techniques. This guideline presents a set of

model-based techniques for the analysis (Section 7.5.2), validation (Section 7.5.3) and verification (Section 7.5.4) of `functional requirements`. Depending on concrete project-specific needs, the quality assurance expert chooses the set of techniques that have to be applied in a specific projects context whenever required or instructed and complements them with instructions for manual reviews.

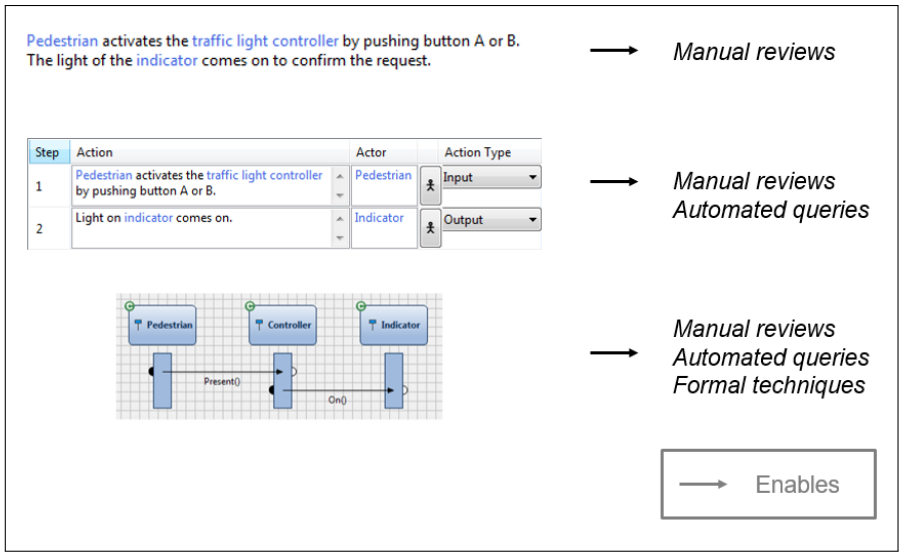

Figure 7.6: Depending on the degree of formalization of a `functional requirement`, the quality assurance expert can choose from a set of quality assurance techniques

7.5.1 Perform Quality Assurance

The quality assurance expert selects the checks to be performed on the requirements specification.

Preconditions. `Functional requirements` are specified.

Outcome. The `functional requirements` are checked.

Steps. For each `functional requirement`, the quality assurance expert created a `QA checklist`. The quality assurance expert defines the *QA check items*, the *author* (person or automated technique) that conducts a check and the *action* to be performed. The action should include the precondition, under which the check is to be performed. For example, a `functional requirement` may only be validated after it is analyzed.

For each *QA check item*, the author performs the quality assurance check that the quality assurance expert assigned, documents the *result* and sets the *QA check status*. Inconsistencies between requirements are documented as `conflict links`.

Tool Support. A tool can facilitate to store and manage the `QA checklists`.

7.5.2 Analyze Functional Requirements

The quality assurance expert performs the requirements analysis to reduce the number of quality issues in the requirements specification. The *status* of all successfully analyzed `functional requirements` is set to *Analyzed*.

7.5.2.1 Check Conformance to the MIRA Artifact Reference Structure

Conformance constraints check whether the requirements specification conforms to the MIRA artifact reference structure. This conformance ensures that a requirements specification has the characteristics with a positive impact on model-based quality assurance as discussed in Chapter 4.

Preconditions. The requirements specification is documented.

Outcome. The requirements specification is conformant to the MIRA artifact reference structure.

Checks. Completeness constraints [Sch09, p. 51] enforce that certain elements are present in the requirements specification in order to be considered as conformant to the MIRA artifact reference structure. The quality assurance expert could conduct these checks before defining the `QA checklists`.

- *Defined RE concepts.* The MIRA artifact reference structure defines a set of RE concepts. Each of these RE concepts should be instantiated in the requirements specification. An exception are trace links; constraints on trace links are defined below.
- *Defined Attributes.* Each mandatory attribute of an RE concept should be defined. If the attribute is not applicable in the project, the requirements engineer can either tailor the artifact reference structure, or document "N/A" (not applicable) in that field.

Consistency constraints [Sch09, p. 52] require that the elements obey specific relationships in a requirements specification to be considered conformant to the MIRA artifact reference structure.

- *Defined Refinement Links.* Each `functional requirement` should be refined from a `requirement`.
- *Defined Realization Links.* Each `functional requirement` should either refine to another `functional requirement`, documented in a `refinement link`, or should be realized in a subsequent development artifact, documented in a `realization link`.
- *Defined Cross-references to Terms.* The *source* of each `functional requirement` should be contained in the list of `requirement sources`. Domain-specific terms used in `functional requirement` should be defined as a `descriptive element`, as a `glossary term` or a `requirement`

source. Domain-specific terms of functional requirements comprise at least the *design scope*, *stimuli* and *responses*. Each of the descriptive elements should include a *definition*.

- *Defined Cross-references to Actors.* Each *actor* defined in a scenario or user function should reference to a requirement source.
- *Scenario Consistency.* The *action types* of a scenario can be used for quality assurance: An *external action* should not imply requirements on the system. *Input actions* and *output actions* should be refined by a functional requirement. Following good RE practices, requirements should always describe a desired effect in the environment [ZJ97]. Therefore, *internal actions* should be checked whether they have to be included in a scenario.
- *User Function Consistency [VEH+14].* A *scenario* of a user function should start with the precondition and trigger as stated by the user function. Every user function should have at least one *success scenario* that ends with the *success guarantee*. If *success guarantee* and *minimal guarantee* differ, at least one *failure scenario* should end with the *minimal guarantee*. The list of *actors* defined in the user function should be consistent with the *actors* in the *stimuli* and *responses*. The information contained in an *input action* should be consistent with the list of *stimuli* of a user function and correspondingly the *output action* with the list of *responses*.

Tool Support. Many conformance checks can be automated as simple database queries on the MIRA artifact reference structure.

7.5.2.2 Perform a Formal Analysis on Functional Requirements

A formal analysis uses the formal representation of functional requirements and their refinement links.

Preconditions. The functional requirements, respectively its refinement link is formalized.

Outcome. Formal analysis can detect quality issues.

Checks. A state automaton that represents a user function can be analyzed with respect to a set of quality factors:

- *Non-determinism check.* A non-determinism check analyzes whether a user function has a deterministic behavior. A non-determinism indicates missing interface requirements.
- *Unreachable states.* An unreachable state of a user function may indicate missing or inconsistent interface requirements.

The formal refinement links of a functional requirements can be analyzed for missing functional requirements and refinement links:

- Each *stimulus* or *response* (modeled as input ports or output ports) of a `functional requirement` should be refined to a *stimulus* or *response* of a refined requirement.
- Each *stimulus* or *response* of a `functional requirement` should originate from a *stimulus* or *response* of a refining `functional requirement`.

Tool Support. A tool can support the efficient application of formal analysis techniques by automation. For example, the list of analysis techniques for `user functions` is an overview of the formal analyses enabled by the tool AutoFOCUS3.

7.5.3 Validate the Functional Requirements

The validation ensures that the requirements specification corresponds to the actual stakeholder needs. Therefore, a set of representative reviewers have to examine the requirements specification. The validation is performed on all `functional requirements` that have the *status Analyzed*. A means to support the validation is the simulation of executable models. After the validation, the *status* of all `functional requirements` that pass the validation is set to *Validated*. Alternatively, requirements can be rejected by the stakeholders and thereby marked with the corresponding *status Rejected*. Only validated requirements have to be considered in the subsequent development activities and are subject to verification and testing.

7.5.3.1 Consolidate the Context Information Used in the Functional Requirements

Ensure that all stakeholders have the same understanding of the context information used in the functional requirements.

Preconditions. `Glossary terms` and `requirement sources` have the *status New*.

Outcome. The set of `glossary terms` and `requirement sources` is consolidated by the stakeholders.

Steps.

1. Select all `glossary terms` and `requirement sources` that are used in the `functional requirements` that are to be validated.
2. Set these `glossary terms` and `requirement sources` to *In Consolidation*.
3. Stakeholders have to agree on the *definition* of each term, or the *definition* has to change accordingly. Deviations in the meaning of a term have to be documented.
4. Stakeholders have to confirm synonyms, abbreviations and other documented information.

5. If all stakeholders agree on the description of the `glossary terms` and `requirement sources`, set its *status* to *Consolidated*.

7.5.3.2 Validate Functional Requirements

Ensure that the refinement from stakeholder `goals` to `functional requirements` conforms to the expectations of the stakeholders.

Preconditions. Textual specification performed as defined in Section 7.3. The set of `functional requirements` to be validated has the *status Analyzed*.

Outcome. The `requirements` are validated and therefore set to the *status Validated*.

Steps.

1. Select all `goals` that are refined to the set of `functional requirements` under validation. The stakeholders have to agree on each of the `goals`. When the stakeholders agree, set their *status* to *Validated*.
2. Select all `refinement links` from the `goals` to the set of `functional requirements`. The stakeholders have to agree on the `refinement links` and the `functional requirements`. When the stakeholders agree, set the *status* of the `functional requirements` to `Validated` and the `refinement links` to *Consolidated*.

Tool Support. A tool could support the stakeholders in the validation by selecting the `goals` that are refined to a set of `functional requirements` and to visualize the `refinement links`. Furthermore, a tool could provide access to all `requirement sources` and to the definition of the domain-specific `glossary terms` during the validation. The tool could simulate those `functional requirements` that are represented as an executable model.

7.5.4 Verify Functional Requirements

Verification ensures that the `functional requirements` are implemented correctly in the subsequent development artifacts.

In the model-based verification, both the `functional requirements` and subsequent development artifacts are represented as formal models. Model-based verification can detect two kinds of fault: *Engineering faults* are contained in a model due to design decisions of an engineer who develops the system. *Modeling faults* occur, when a model does not reflect the intentions of an engineer and can occur both in the model of a requirement or another development artifact. Identifying an engineering fault gives the engineer the possibility to revise the underlying design decision and

correct it. When identifying a fault, the corresponding model needs to be corrected and the verification has to be rerun.

A syntactic verification of a `refinement link` between `functional requirements` or a `realization link` from a `functional requirement` to a design model is achieved by modeling a formal refinement specification. A semantic verification can be done using a formal verification technique. After a successful verification, the *status* of the `functional requirement` is set to *Verified*.

Tool Support. A prerequisite for applying model-based verification efficiently is a tool that supports the formal modeling notations including the formal refinement specification and the formal techniques. Further tool support can be provided for the analysis of the results in order to identify a fault. AutoFOCUS3 provides three formal techniques, MSC conformity check, formal verification of I/O assertions, and test case generation and execution. These three techniques are presented in the following.

7.5.4.1 Perform Formal Verification of I/O Assertions

Ensure that a development artifact (a `user function` or a model of the functional or logical viewpoint) that is represented as a component model conforms to an `interface requirement` that is represented as an I/O assertion.

Preconditions. The `interface requirement` is represented as an I/O assertion. The `interface requirement` is refined or realized by a development artifact that is modeled as a component model, that means, a (hierarchical) component with specified interface behavior. If the I/O assertion is a subset of the syntactic interface of the component model, the component model can be verified directly to check that it fulfills the I/O assertion. If the I/O assertion is not specified at the same level of detail as the component model, a formal refinement specification is necessary that maps the syntactic interfaces of I/O assertion and component model.

Outcome. The formal verification either confirms that the I/O assertion is fulfilled by the design model or it detects a fault. If the formal verification detects a fault, it generates a counter example that can be simulated and thereby analyzed.

Figure 7.7 shows the synthesis of `interface requirements` into a `user function` and the verification of the synthesis. The syntactic interface of Requirement 1 is Input1, Input2, Output3. The syntactic interface of Requirement 2 is Input4, Output3. Both syntactic interfaces are a subset of the syntactic interface of the user function Requirement1_2_x that is Input1, Input2, Input4, Output3. The verification result is 'SUCCESS'. This means that the I/O assertions both hold in the state automaton of Requirement1_2_x. The verification can be seen as a proof, if the verification is not constraint by some boundary.

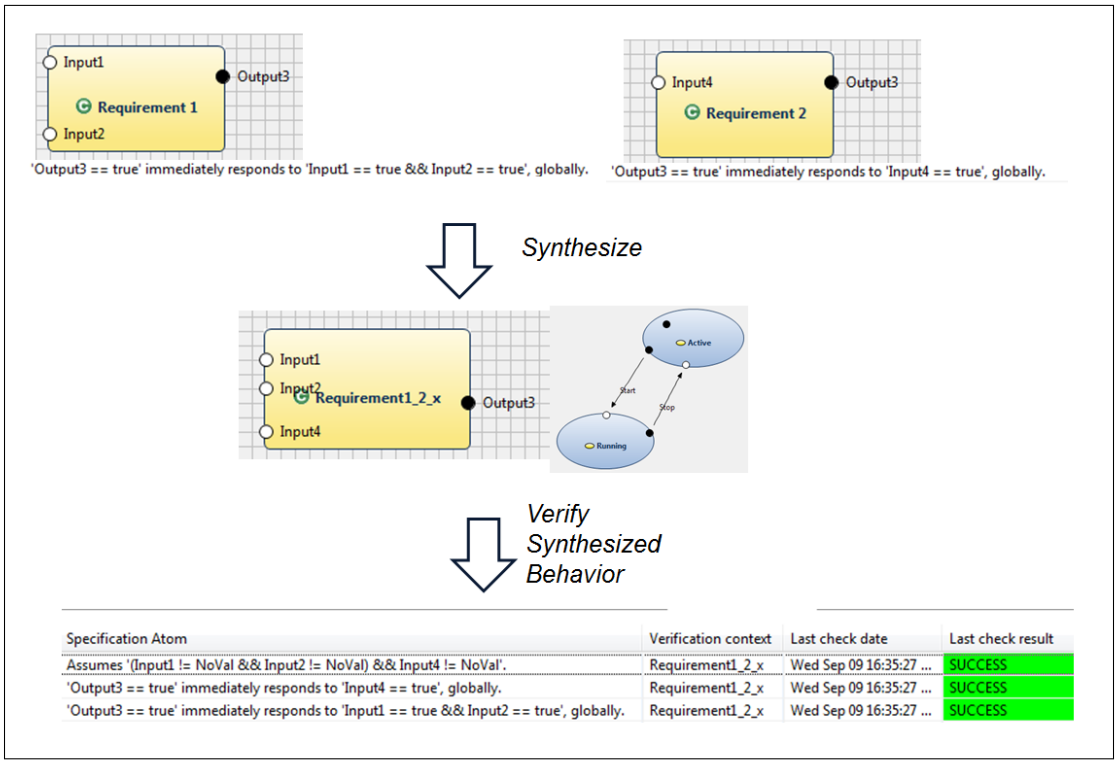

Figure 7.7: Manual synthesis and formal verification of two `interface requirements` in a `user function`

7.5.4.2 Perform MSC Conformity Check

The MSC Conformity Check checks that the sequence of messages defined in a `scenario` or `user function` that is represented as an MSC is executable in a development artifact that is represented as a component model, for example, in a `user function`, a model of the functional or logical viewpoint.

Preconditions. The `scenario` or `user function` is represented as an MSC. The development artifact is represented as a component model. The syntactic interface defined in the MSC is a subset of the syntactic interface of the component model.

Outcome. The MSC conformity check either confirms that the MSC is executable in the model of the development artifact, or it detects a fault. If the check is successful, the tool provides an example that can be analyzed.

7.5.4.3 Generate and Execute Test Cases

Test that a development artifact (a `user function`, a model of the functional or logical viewpoint) represented as a component model is conforming to a given `user function` represented as a component model. It checks that the sequence of messages defined in an MSC is executable in the development artifact.

Preconditions. The `user function` is represented as a component model. The `user function` is refined or realized by a development artifact that is represented as a component model. A formal refinement specification formalizes the `refinement link` or `realization link`.

Outcome. The execution of the test cases checks, whether the input-output behavior defined in the `user function` can be fulfilled in the development artifact. The coverage of the test cases is pre-set in coverage criteria.

7.6 Summary and Discussion

The major activities contained in the MIRA guideline have been taken from three guidelines, see Section 7.1, and adapted to the MIRA artifact reference structure and to the quality assurance techniques provided by AutoFOCUS3. The MIRA guideline describes the textual specification and the formalization of functional requirements on the MIRA artifact reference structure. Furthermore, a set of model-based quality assurance activities for the analysis, validation and verification are presented. The guideline shows that the MIRA artifact reference structure is sufficient to support these quality assurance activities. The quality assurance expert can choose and tailor those activities that are relevant for the projects context. Defining the MIRA guideline revealed the following topics that constitute open research questions.

7.6.1 The Ratio of Requirements to Formal Models

MIRA formalizes each requirement independently from other requirements in one or more formal specifications. The ratio of requirement to formal models here is 1:n. In a second step, formal requirements may be synthesized into a state automaton. The ratio of requirements to state machines is m:n.

Other approaches such as ADORA [GBR+00] integrate a set of requirements (or other contents of the requirement specification) directly into one or more common models. An ADORA model integrates all modeling aspects (for example, structure, data, behavior, user interaction) in one coherent model. The ratio of textual requirements to this model then is n:1.

Both approaches have advantages and disadvantages. The independent formalization of requirements as propagated in MIRA has the advantage to decrease model complexity: A model that represents a requirement or even only parts of a requirement contains less model elements than a model that integrates several of these requirements. This low number of model elements simplifies to verify whether the model corresponds to the textual requirement. Also verifying the state automaton that has been synthesized from I/O assertions becomes less difficult, as it does not have to be checked manually; the state automaton can be verified automatically using formal verification techniques to check whether it fulfills all I/O assertions. A formalization of requirements independently from other requirements has a second advantage. The requirements can be formalized in an incremental manner. If a requirement is not formalized, then the quality of the models does not suffer.

Following [GBR+00], a coherent model over all requirements allows to "develop a strong notion of consistency and provides the necessary basis for developing powerful consistency checking mechanisms in tools". They claim that the model construction becomes more systematic, reduces redundancy and simplifies completeness checking.

In SPES and FOCUS, an integration of functional requirements into a coherent model is not part of the requirements viewpoint. In these approaches, the functional viewpoint provides a coherent model of the required system behavior. The reason for separating the requirements viewpoint and the functional viewpoint is the necessity of architectural decisions that are taken in the functional viewpoint. These architectural decisions are for example on the functions the system should offer, the interfaces of these functions and the dependencies between these functions.

A detailed comparison of advantages and disadvantages of independent models compared to coherent models in the requirements viewpoint has to be investigated further and remains an open research question.

7.6.2 Redundancy in the Representations of a Requirement

A requirement can be represented as an informal description, as structured text or formally. A step-wise formalization of a requirement that includes either all three representations or from structured text to the formal representation has been propagated in many approaches, for example in [CRST09] or [KC05a]. A requirement represented in more than one representation contains redundant information. How to deal with this redundancy has to be resolved before a project starts.

Redundancy in the formal representation was discussed by Heimdahl [Hei07]. In the discussed approach [MTWH06], analogously to MIRA, natural language requirements, temporal logic and an executable formal model were documented. He argues that formal models of any substantial system are likely to be incorrect with respect to the real needs of the system. To obtain correctness, the models have to be checked against each other in a rigorous validation process. In this approach, documenting natural language requirements and temporal logic requirements, and not only the executable models, detected significant flaws in the requirements. Modeling the executable model helped to improve the understanding of the real needs of his customers.

Redundant information can be avoided for example by deleting the less formal representation, as soon as the information is represented more formally. Another possibility to resolve this redundancy problem is to declare the more formal representation as the *single source of truth*. Depending on the domain and project-specific settings, the information has to be stored redundantly as informal text, structured text and in the formal representation. If more than one representation of a requirement is documented, the consistency between the representations must be ensured each time after changing one of these representation forms. The most efficient solution for a specific domain is an open issue that requires further investigation.

7.6.3 Formal Refinement Specifications for Verification

The specification of a formal refinement specification can be a complex task as a case study [TBP14] confirmed. In this case study, a formal refinement specification was used to explicitly document the refinement from high level requirements to more detailed requirements. The case study indicates that the formal refinement specification is straightforward for a refinement of the required system interface and system behavior, but it can become quite difficult in other cases. The case study investigates the difficulties of specifying a formal refinement that includes both a refinement of the interface behavior and a decomposition of a system into subsystems. In this case, the high level requirement is described at the system level; the detailed requirement not only describes a more detailed system behavior, but describes this refined behavior on a subsystem level. The MIRA approach recommends to set up refinement specifications only between artifacts with the same design scope.

Many approaches such as [RRH93] avoid the difficulties of the specification of a formal refinement by proposing to formalize requirements at the same level of detail as the subsequent design models. This means that the formal representation of a requirement uses the syntactical interface of the design. This implies explicit knowledge about the system design already during the formalization of requirements. It also means that the formalization of the requirements has to be performed after building the design model. As a consequence, such a late formalization impedes the use of formal analysis techniques before their implementation. Flaws can only be identified after building the design model.

An alternative to setting up a formal refinement specification and to a late formalization would be to have two formal representations of the requirement. An early formalization that is independent of the design enables formal analysis techniques. Formalization at the design level enables formal verification. This redundant specification would lead to an increased specification effort. Further investigations are required in order to determine the most efficient solution.

Chapter 8

The MIRA Tool

The MIRA tool implements the MIRA artifact reference structure that was presented in Chapter 6 and provides operations on the artifact reference structure. These operations are derived from an initial investigation of essential characteristics of the MIRA approach presented in Chapter 5, from a systematic investigation of the MIRA guideline that is introduced in Chapter 7, and from feedback of the MIRA users that applied MIRA in various case studies that are presented in Chapter 9.

The research question addressed in this chapter is:

> ***RQ5:*** *What operations on the MIRA artifact reference structure are necessary in a tool for conducting the MIRA guideline?*

The main contribution of this chapter is the demonstration of how the MIRA tool implements the MIRA artifact reference structure and supports the MIRA guideline. The implementation of the MIRA approach shows that it is technically feasible to implement the MIRA artifact reference structure.

The context of the implementation of the MIRA tool is given in Section 8.1. The MIRA tool is presented in Section 8.2. Section 8.3 summarizes the chapter.

Contents

8.1 Implementation Context

The MIRA tool [TMR13, AVT+15] is implemented as an extension to the computer-aided development tool AutoFOCUS3[1]. AutoFOCUS3 is presented in detail in Chapter 2. The integration in AutoFOCUS3 yields several benefits. MIRA adds requirements engineering capabilities to AutoFOCUS3. Therefore, AutoFOCUS3 benefits

[1] http://af3.fortiss.org

from the integration, since with the MIRA tool "seamless model-based development" [BFH+10] from requirements to code generation can be provided by AutoFOCUS3. The AutoFOCUS3 tool provides an implementation of the formal system modeling theory FOCUS and a set of complementary formal quality assurance techniques. Through MIRA, these techniques can be applied in the RE context.

The MIRA tool has been implemented in a collaboration of the author with Dongyue Mou and Daniel Ratiu. Dongyue Mou was responsible for the architecture of the MIRA tool, and developed formal refinement specifications and requirements-based testing in AutoFOCUS3. Daniel Ratiu contributed the formal representation of requirements and user-friendly formal quality assurance techniques to the MIRA tool.

A short overview of the used technologies: The MIRA tool is implemented as a plug-in to AutoFOCUS3. AutoFOCUS3 provides an elaborated framework[2] for the development of a model-based development tool based on Java/Eclipse. The MIRA artifact reference structure was instantiated as a data model using the Eclipse Modeling Framework (EMF)[3] model. The user interface was implemented using the Standard Widget Toolkit (SWT)[4].

8.2 Implementation of the MIRA Tool

The MIRA tool implements the MIRA artifact reference structure as presented in Chapter 6. To document the system context, the MIRA tool implements `glossary terms`, `requirement sources` (`documents`, `external systems`, and `stakeholders`).

On the requirements level, the MIRA tool implements generic `requirements`, `user functions` and `scenarios` as use cases and `interface requirements` as interface behavior requirements. `Goals` can be documented as generic requirements in the MIRA tool. For every `requirement`, the MIRA tool offers `QA check lists`.

The MIRA tool facilitates to document `refinement links`, `conflict links`, `external trace links`, and `actor - requirements source` cross references. Cross references for `term definitions` are generated by MIRA. Figure 8.1 gives an overview of the concepts implemented in MIRA.

8.2.1 Textual Specification

The MIRA artifact reference structure defines a set of concepts and their attributes. These concepts are visualized and edited through templates. Each template is divided into sections that group the displayed information.

Glossary Terms and Requirement Sources. Glossary terms and requirement sources are descriptive elements. Each template for a descriptive element consists of

[2] https://af3-developer.fortiss.org/projects/autofocus3/wiki/Tooling_Kernel_Concepts_and_Services
[3] https://eclipse.org/modeling/emf/
[4] http://www.eclipse.org/swt/

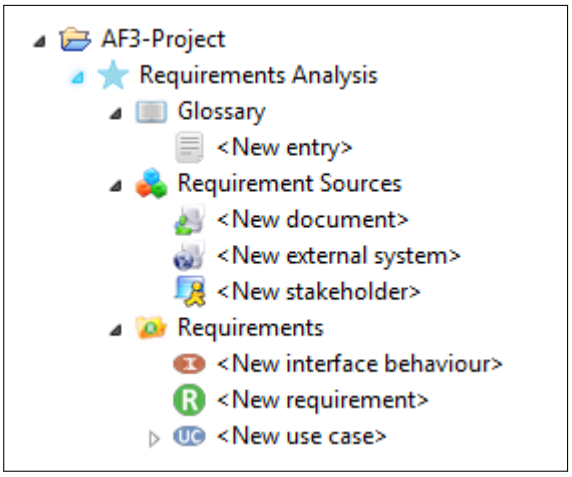

Figure 8.1: An example menu with all concepts that are implemented in the MIRA tool

four sections: The *general* section defines the name, definition and status of a descriptive element. The *synonyms* section contains a list of synonyms, and the *abbreviations* section a list of abbreviations. The last section is to document a *comment*. Glossary terms as shown in Figure 8.2, documents, stakeholders, and external systems use the generic template for descriptive elements and extend it with additional fields as defined in the artifact reference structure.

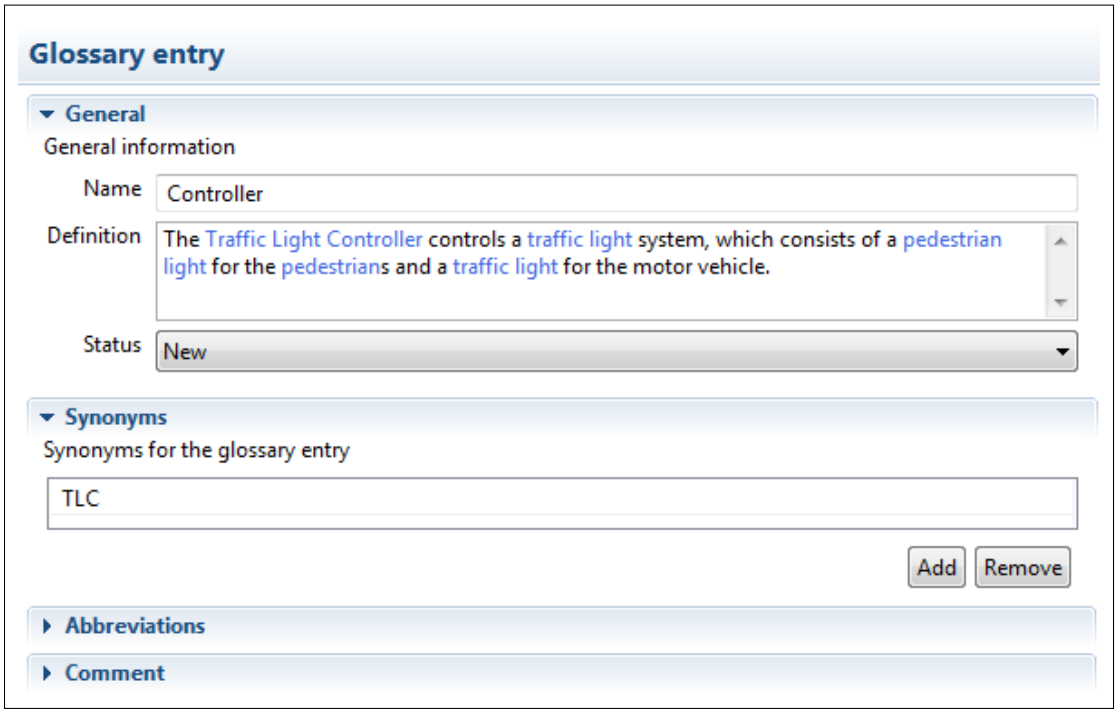

Figure 8.2: Template for a glossary term

Requirements. The MIRA tool provides a template for generic requirements. The requirements template can be used to document all those requirements where the

MIRA tool does not implement a specific template. Figure 8.3 shows an example where the requirements template is used to document a goal. Requirements management attributes and the informal description of the requirement are bundled in the *general* section. Informal descriptions of a requirement may be supplemented with *images*. Relations between requirements and to architecture are managed in the *traces* section. The last section is to document a *comment*.

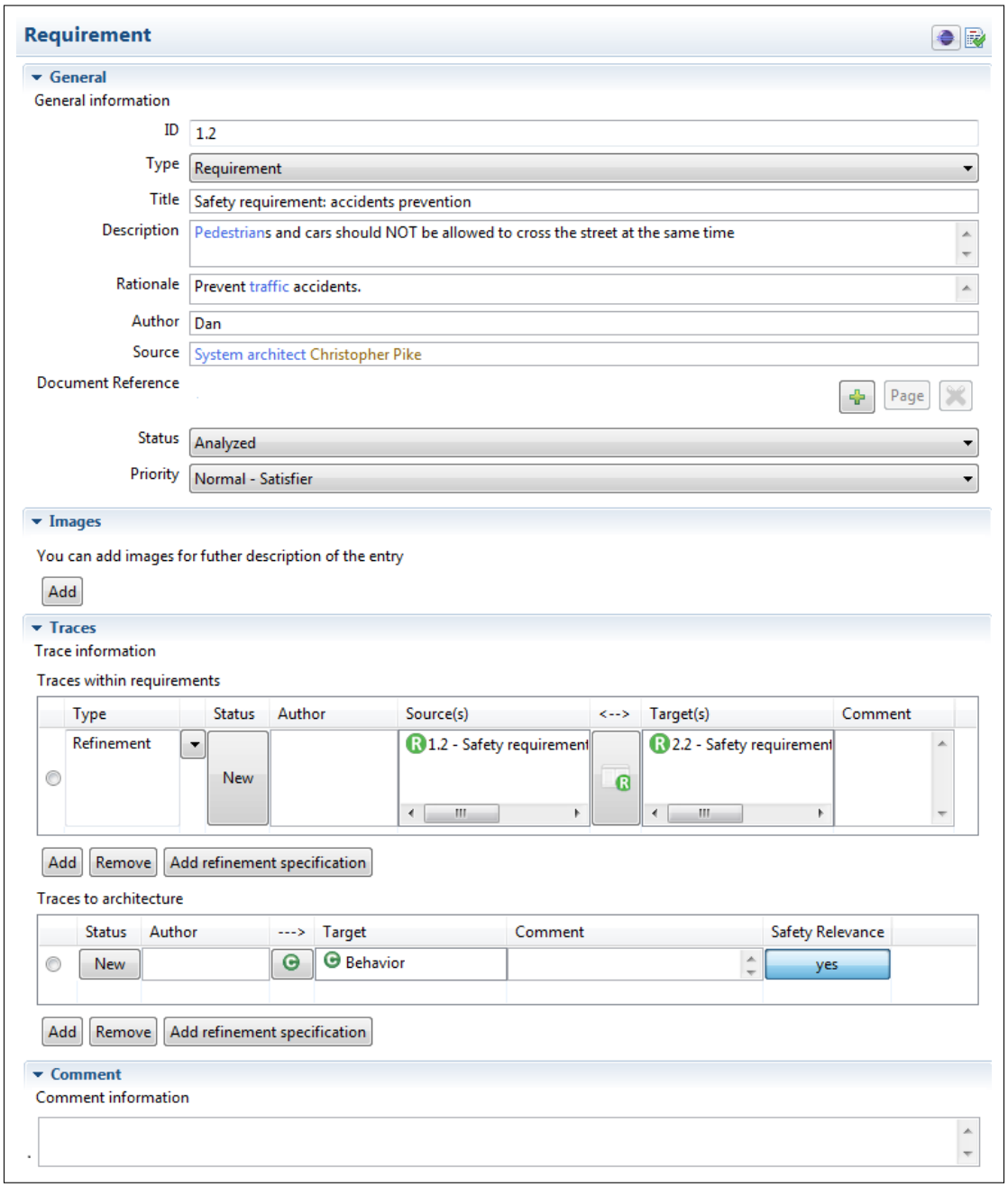

Figure 8.3: Template for requirements

Templates for use cases and interface behavior requirements are specializations from the generic requirements template introduced above. Both templates include all sections with all attributes from the requirements template. In addition to the require-

ments template, a template for a specific type of requirement can provide additional attributes in an optional *detail* section. The *detail* section of the use case template is shown in Figure 8.4. Use cases can be further described by scenarios, see Figure 8.5. The *detail* section of an interface behavior requirement template is given in Figure 8.6.

Figure 8.4: *Detail* section of a use case template

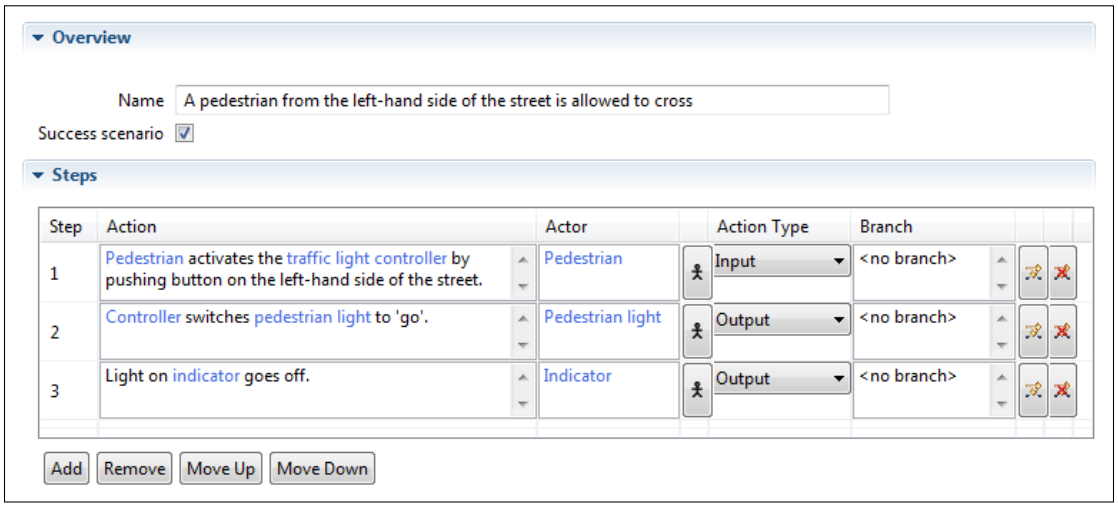

Figure 8.5: Template for a scenario

Categorization of Requirements. The type of a requirement can be changed using a drop-down menu in the field *type*. As a result, the type has the new value and the *detail* section for the chosen type is created. This operation supports the categorization of requirements: Before the requirements type is determined, the user documents the requirements as a generic requirement. When the user classified the requirement, then the user can change the requirements type, for example, to a use case. Thereby, the tool changes the type to use case and adds the *detail* section of a use case that is shown in Figure 8.4.

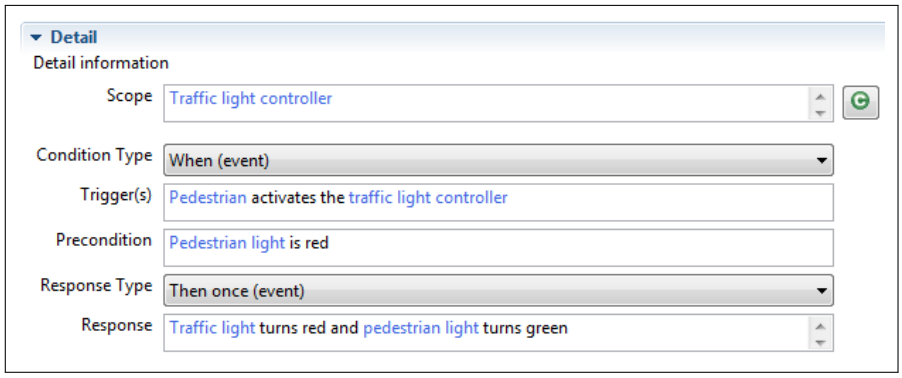

Figure 8.6: *Detail* section of an interface behavior requirement template

Traceability of Context Elements. MIRA offers a set of operations to create and access terms of the system context in order to reduce the risk of lexical ambiguity. MIRA supports the designation of terms: The user can select a term, for example, from the requirements template, and add it to the context elements by right-clicking on that term. Then the user chooses whether to create a glossary term or to add a requirements source. This choice creates the template for the selected term. In the template, the user now can add the term definition, synonyms and abbreviations. The correct use of context elements is supported by auto-completion. MIRA offers operations that guarantee a quick access to the terms: MIRA highlights all terms that are documented as context elements. MIRA highlights the name of a context element and its abbreviations in blue. Synonyms are highlighted in red. When a user moves the mouse over a term, an instant display as a pop-up facilitates a quick access to the definition of a term. For example, the term 'pedestrian' is defined in the glossary and used in a requirement and therefore highlighted. A pop-up gives the definition of 'pedestrian', see Figure 8.7. Clicking on the highlighted term in the pop-up opens the object 'pedestrian'. The links between context elements and their use are generated. In some settings, an automated generation of these links might not be sufficient. For example, the glossary can provide two definitions of the same term. To avoid misunderstandings, it might be necessary to ensure that only the correct definition is displayed. Another example is abbreviations that coincidentally correspond to wide-spread terms like 'or'. In these cases, MIRA would have to be extended with technology to explicitly document and manipulate these links.

Traceability of Requirements. The *trace* section of a requirement lists all documented trace links that contain this requirement as a source or target artifact; it lists requirement trace links, i.e., links to other requirements and external trace links, i.e., links from the requirement to a component of a component architecture. Realization links that indicate that a component realizes a requirement are documented as a trace link between requirement and component. Traces to other development artifacts have not been implemented, as they were not considered to be relevant for functional requirements in the course of our studies. Besides the template, the MIRA tool implements further support for the specification and quality assurance of trace links: The MIRA artifact reference structure facilitates predefined queries. A query can pre-select a set of potential sources and targets of a trace link. The user then can

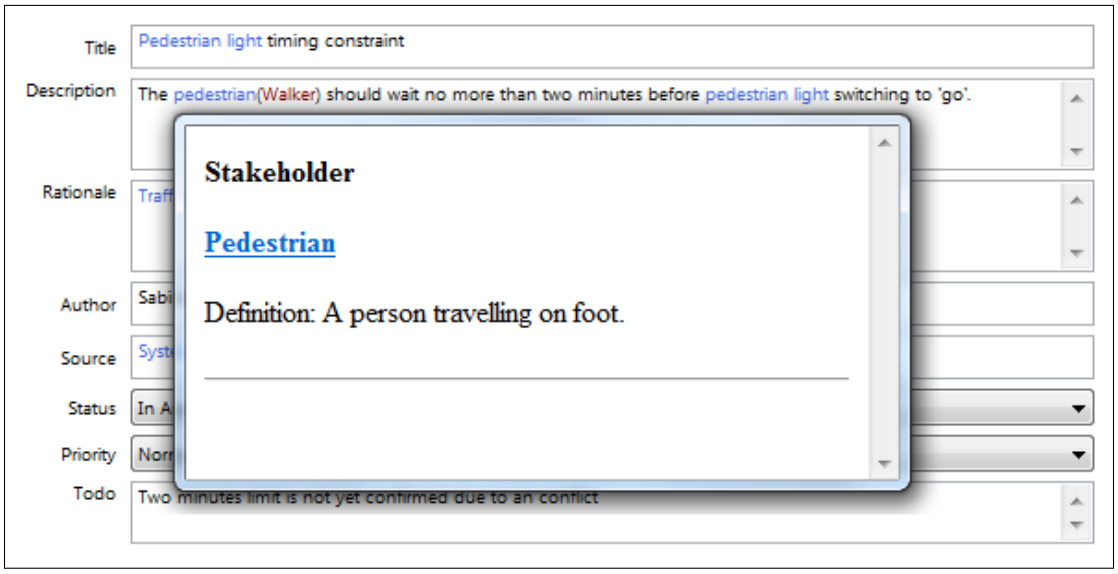

Figure 8.7: When holding the mouse cursor on a highlighted term, a pop-up shows additional context information

choose the sources and targets from the pre-selected artifacts. For example, when the user creates an external trace link from a requirement, MIRA pre-selects the components from the component architecture as a target. Other objects, for example, other requirements, are not shown and cannot be selected. This pre-selection supports the specification of trace links. Requirement trace links and external trace links can be visualized. MIRA offers a hierarchical view of the requirement trace links, see Figure 8.8. This view supports the quality assurance of trace links and can be used for example in an impact analysis of requirements.

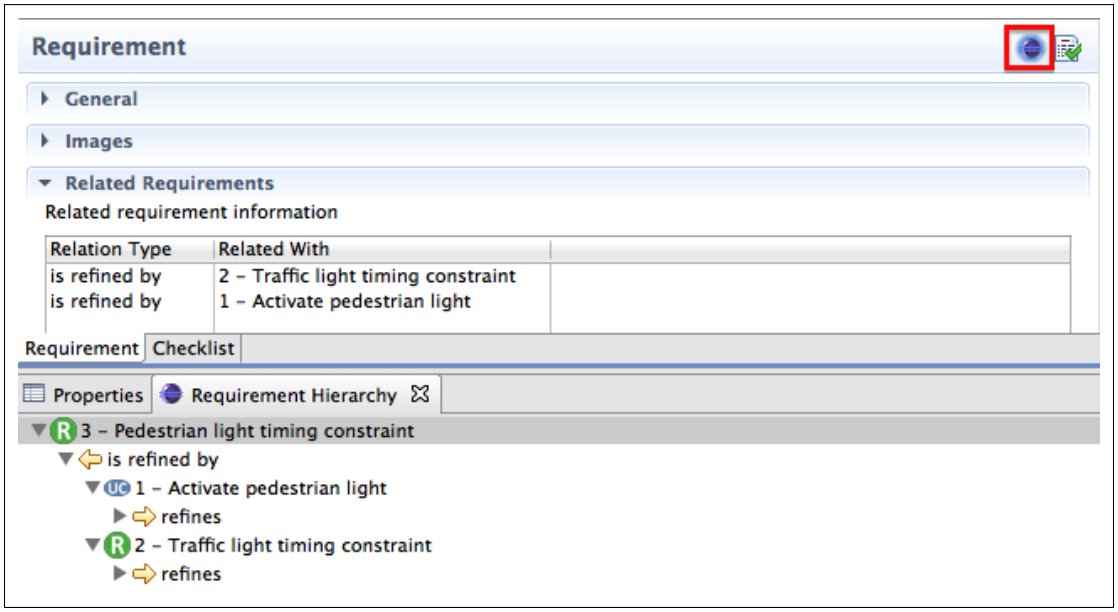

Figure 8.8: The requirements hierarchy visualizes trace links between requirements

Actor – Requirement Source Cross References. The MIRA artifact reference structure defines a cross reference between requirement sources and use case. Every actor in a scenario step of a use case has to be defined as a requirement source. Therefore, actors have to be chosen from the list of requirement sources or a new requirement source has to be created. Figure 8.9 shows the selection dialog in the foreground and the edited scenario in the background. On the left side of the selection dialog all documented requirement sources are displayed. From this list, the requirement source can be selected and used as an actor of a scenario step by dragging them to the right. With the buttons on the left border of the selection dialog, new requirement sources can be added or existing sources can be edited. This mechanism ensures that actors are listed in the requirement sources and are considered in other RE activities, for example the elicitation of requirements.

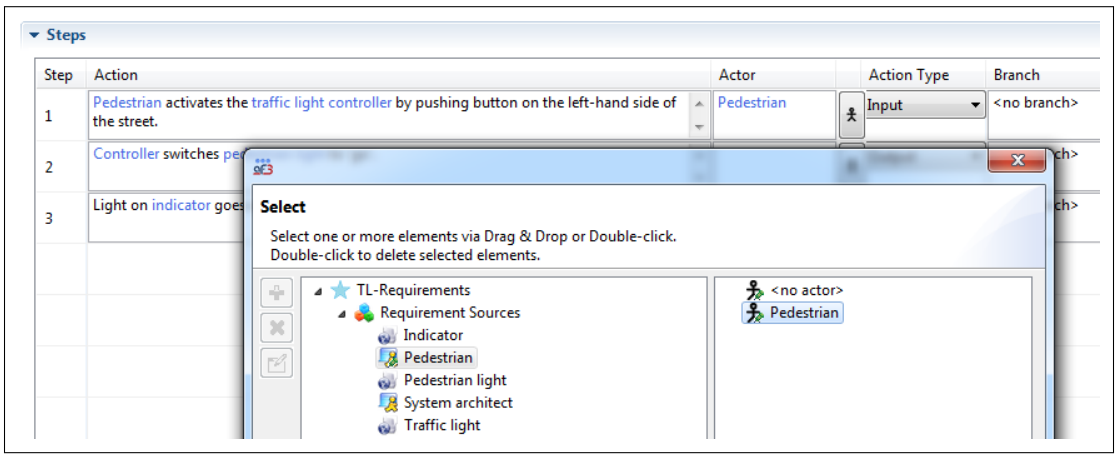

Figure 8.9: The actor of a scenario step is selected from the list of requirement sources

Change Support. MIRA offers operations that support the user in the case of changes to the requirements specification. These operations are based on the predefinition of trace links in the MIRA artifact reference structure. MIRA propagates changes in the ID or title of a requirement immediately to the requirement trace links and external trace links of this requirement. MIRA applies and visualizes changes in the source and target artifacts of a trace link immediately in the trace sections of all concerned requirements. MIRA offers a simple copy-and-paste operation for context elements and requirements to support changes. When a user copies a requirement, MIRA does not copy the requirement trace links and external trace links of this requirement, but it informs the user that these links have to be recreated if necessary. When the user changes a glossary term or the name of a requirements source, he/she should document the previous name in the synonym list. Then the user can identify outdated terms manually, as MIRA highlights each synonym in red.

Advanced Overview and Search. A model-based approach facilitates advanced overview and search operations. The objects of the requirements specification can be displayed in an overview. In MIRA, this overview is available on several levels:

- All objects contained in the requirements specification;

- All objects in a package (a package groups a set of glossary terms, requirement sources or requirements);
- All occurrences of a name, synonym or abbreviation of a context element in other context elements or requirements;
- All trace links.

This overview provides a model-based search that filters and sorts by concepts and object attributes. To enhance this capability, the model-based search can be combined with text-based search. To demonstrate this, MIRA includes a search for similar terms. Statistics provide a quick overview on the number of documented objects. The model-based overview and search was used in the course of the case studies to analyze requirements, to trace the current quality assurance status of objects and to create trace links.

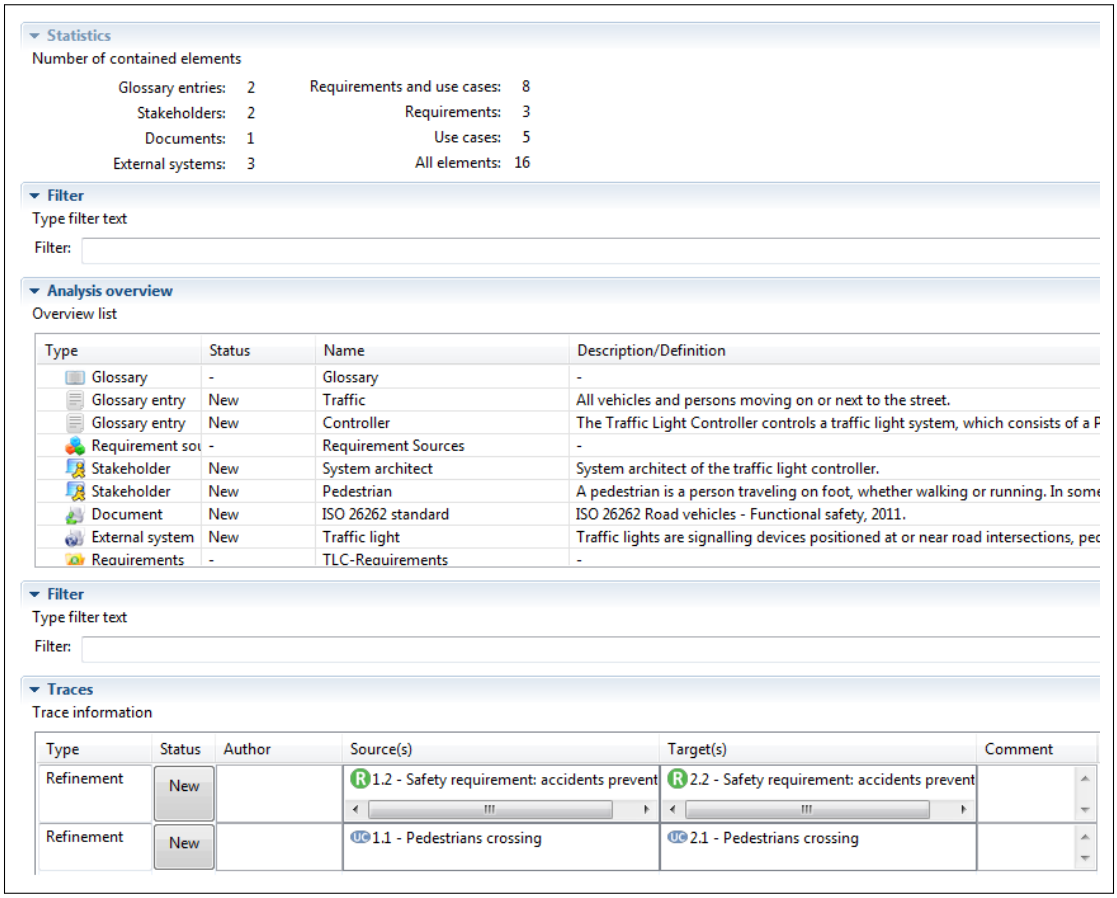

Figure 8.10: An overview and search dialog

Navigation by Cross References. Whenever possible, objects are referenced instead of documented as prose. In MIRA, every object that is referenced is also navigable. Clicking on the reference of an object opens the object itself. To give two examples: Search results are a list of references. The source and target objects of a trace link are references. Documenting a trace link therefore automatically facilitates navigation to the traced objects.

8.2.2 Formal Specification

A data dictionary is used in AutoFOCUS3 in order to document data types and their values. The formal specification of requirements is attached directly to each requirement. In Figure 8.11, the menu of a use case and a requirement with their formal representation is shown. The tool can strictly enforce the application of the guideline with respect to the allowed modeling languages: Depending on the requirement type, only certain modeling notations are available. To provide an example, In the MIRA tool, only use cases can be structured by scenarios and formalized by MSCs. Interface behavior requirements cannot be formalized by MSCs. The different selection dialogs are shown in Figure 8.12.

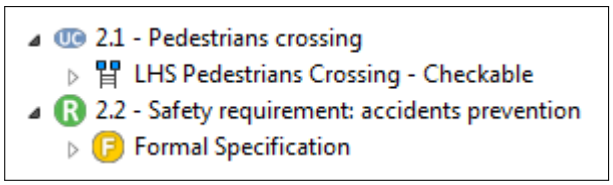

Figure 8.11: Formal representation for use cases and requirements

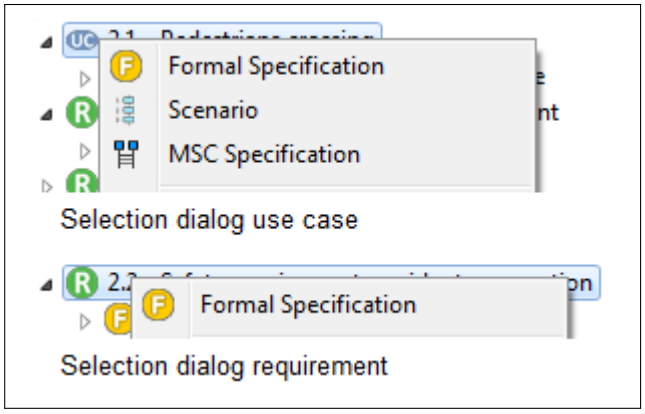

Figure 8.12: Selection dialog for use case and requirement

8.2.3 Quality Assurance

Overview of the Quality Assurance Status. In MIRA, each object has a quality assurance status. The search dialog presented in Figure 8.10 displays the status of each object and facilitates to filter and sort the objects according to their status. This gives a quick access on the current quality assurance status.

Manual Reviews. For each requirement, user-defined and project-specific checklists for manual reviews and their results can be created, see Figure 8.13.

Automated Reviews. The MIRA artifact reference structure facilitates predefine checks on the requirements specification through database queries for an automatic

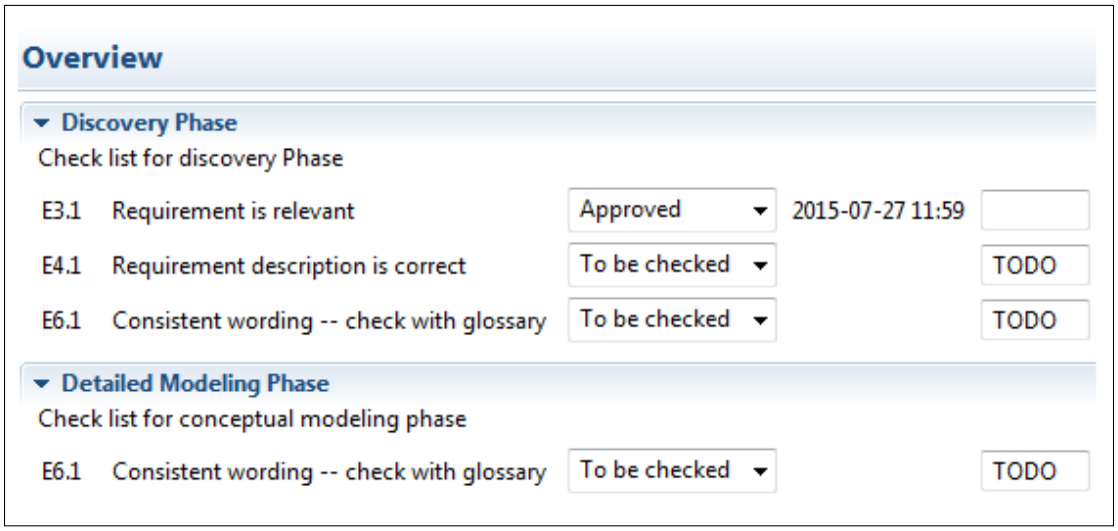

Figure 8.13: Manual review checklist and results

review. Syntactic checks inspect whether templates are filled out completely and point out missing information. For example, a syntactic check of a requirement may reveal an empty author field and a missing description. By contrast, to-do fields are checked to be empty. The second group of checks are simple semantic checks on the values of fields. These checks can identify objects that require further treatment. For example, requirements that are not yet in the status 'analyzed' are marked. The third group of checks analyze trace links. For example, actors defined in a use case have to be used in at least one scenario step of this use case. Use cases and functional requirements have to be either refined to functional requirements or realized by a component. An example for results of such checks are given in Figure 8.14.

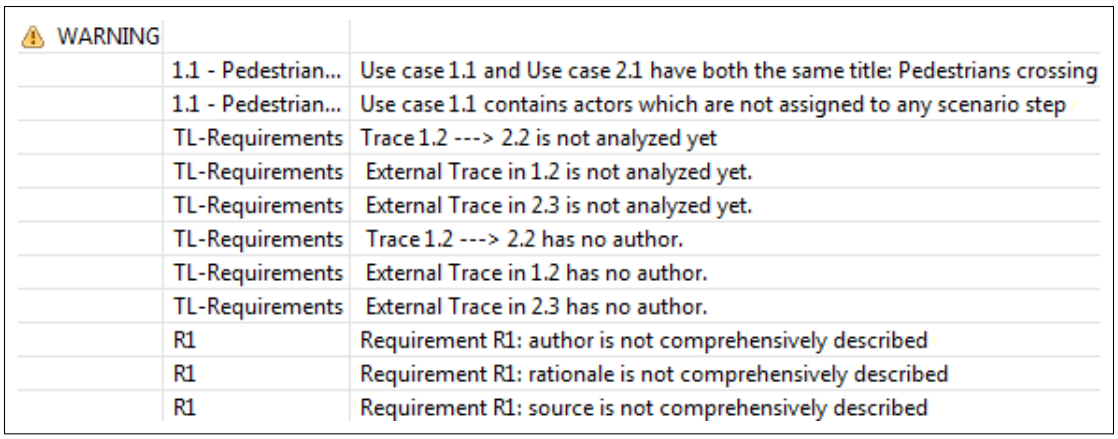

Figure 8.14: Database queries are used to perform automated checks on the data model

Formal Quality Assurance Techniques. The formal specification of requirements and their realization in subsequent development artifacts enables advanced tool-supported analyses. The automation directly impacts the time-efficiency of the analyses. MIRA formalizes scenarios as MSCs, interface requirements as I/O assertions

and user functions as executable models. For these formal representations, AutoFOCUS3 offers the following automated quality assurance methods (see also Section 2.3.6:

Non-determinism check analyzes the transitions of a state automaton. A non-determinism check can identify non-deterministic behavior of a user function modeled as a state automaton. This means that a system may respond differently in the same situation at different times. It should be checked that any non-deterministic behavior is intended and that the cause is not *missing* or *incorrect* requirements.

Unreachable state check can identify unreachable states of a user function modeled as a state automaton. The unreachable state check addresses the quality attribute *logical completeness* and *consistency*. An unreachable state may be caused by missing transitions to that state. In the case that a transition to that state exists, inconsistent requirements may be the cause. Another possibility is that the state and its corresponding requirements are unnecessary and may be removed.

Simulation of an executable model is a common means for the *validation* of requirements [vL09, p. 198 ff]. In AutoFOCUS3, an executable model, i.e. a state automaton or a code specification, can be simulated. MIRA facilitates to simulate user functions that are represented by executable models.

Requirements verification checks that a model of the system under development, here a user function, fulfills a set of interface requirements or scenarios. In AutoFOCUS3, the *verification* can be performed on requirements specified as I/O assertions, assumption/guarantee specifications or MSCs.

Model-based testing (see for example Mou and Ratiu [MR12] or Blech et al. [BMR12]) provides test-suites with pre-defined coverage criteria. In AutoFOCUS3, a test-suite can be generated from a user function modeled as an executable model. It can then be *tested* that an executable model of the system under development, for example in the logical viewpoint, fulfills this test-suite.

8.2.4 Interfaces of MIRA

Document Generation. Practitioners often prefer to have a requirements specification in the form of a document. A rationale for document generation mentioned by practitioners is the validation of requirements. For conducting the validation, stakeholders need access to the requirements specification. Access may be impeded due to costs like licensing, administration and tool-specific training. Some commercial tools provide support for the validation of requirements via a web interface. Nevertheless, in some settings document generation is currently indispensable, for example, when project-specific security issues do not allow the stakeholders to use RE specific tools. An advantage of generating a document is that no additional training is required for stakeholders that are used to standard office software. MIRA implements an export of the requirements specification to the formats 'doc' and 'html'.

Model Export. Exchanging requirements between tools requires an exchange format. A current, wide-spread format is the Requirements Interchange Format (ReqIF) [OMG13]. This format is XML-based. For exporting the complete requirements specification, not only the artifact reference structure, but also the formal specification language has to be mapped to XML. In a seamless development environment such as AutoFOCUS3, a model export is not necessary for system development. However, MIRA supports an export with the ReqIF format.

8.3 Summary

The MIRA tool provides a technical solution for the MIRA artifact reference structure, thereby demonstrating its *feasibility*. The MIRA tool complements the MIRA approach with operations for the specification and quality assurance of functional requirements in accordance to the MIRA artifact reference structure.

Chapter 9

Industrial Case Studies

This chapter reports on the use of MIRA in two case studies in an industrial setting. Both case studies had requirements specifications developed in industry as an input. The first case study answers the research question:

> ***RQ8:*** *Is the MIRA approach effectively applicable in an industrial setting?*

The case study demonstrates that the MIRA approach is applicable for an industrial specification in the domain of train automation. It shows the scalability of the MIRA approach to the size of several user-visible system functions. The case study improved the quality of the industrial specification and additionally resulted in an improved guideline for the MIRA approach.

The main research question that is answered with the second case study is:

> ***RQ9:*** *Is the MIRA approach extensible with project-specific requirement types in an industrial setting?*

The second case study uses MIRA for documenting and tracing a set of quality requirements for a flight control system. The case study confirms the extensibility of the MIRA artifact reference structure with additional requirement types to document quality requirements and trace link types that facilitate the tracing from requirements to the technical viewpoint.

The case studies are presented and discussed based on the structure recommended by Runeson and Höst [RH09].

The case study on the effective application of MIRA in train automation is presented in Section 9.1. The case study on extensibility in the flight control system domain is introduced in Section 9.2. Section 9.3 gives an overview of further case studies that extended or applied MIRA successfully.

Contents

9.1 Case Study on Applicability and Effectiveness

This case study was part of an industrial research project [BJV+14] that was executed in a one-year collaboration between Siemens Rail Automation, fortiss GmbH and the Technical University of Munich. The research objective of the project was to evaluate the applicability of the SPES modeling framework (see Chapter 2) to a real-life productive system in the context of rail automation. In the course of the project, parts of a Siemens train automation system were modeled for the various viewpoints proposed by the SPES modeling framework from requirements to code generation. For modeling the requirements viewpoint, the MIRA approach was used. As tool support for the case study, AutoFOCUS3 was used in its release version 2.6 in which the MIRA tool is implemented as a plug-in. The resulting models for the viewpoints were verified against each other. The benefits and efforts were discussed with Siemens.

This case study reports on the first part of the project, the specification and quality assurance of functional requirements using the MIRA approach including the verification of the models of the functional architecture against the functional requirements. Further information on functional architectures can be found in Chapter 2; further details on constructing the functional architecture in the course of this project can be found in [Vog15, p. 93ff].

The study design of the research project and some of the study results have been previously published in [BJV+14, TBP14].

9.1.1 Study Goal

The goal of this case study was to evaluate the effective applicability of the MIRA approach based on industrial textual specifications for an existing, safety-critical system. Despite the high quality of the input documents, we aim to demonstrate that the application of the MIRA approach leads to an improvement of the quality of these documents.

9.1.2 Study Object

The *Trainguard MT* (TGMT)[1] system is a modern automatic train control system for metros, rapid transit, commuter and light rail systems. The TGMT system provides a large number of protection and automation functions for railway operation and uses components on the wayside and on-board the trains.

TGMT consists of two major subsystems: The *wayside subsystem* calculates the movement authority, the authority of the train to enter and travel through a specific part of track in a given travel direction. The *on-board subsystem* supervises the train operation within the given movement authority limit. The wayside and on-board subsystems use a track database, which stores railway track topography descriptions such as speed and gradient profiles. The on-board subsystem supervises and controls the train movement based on train localization, the information received from the wayside subsystem and the information stored in the track database.

[1] http://sie.ag/1aHfP1J

Figure 9.1: Platform screen doors installed in Paris

The TGMT system concept is based on a cyclical exchange of position report telegrams sent from trains to the wayside subsystem and on movement authority telegrams sent from the wayside subsystem to the trains. Telegrams are standardized records that are transmitted digitally and used for control purposes of the system.

The project scope was to model the *Platform Screen Doors* (PSD) function of the TGMT system. PSDs are installed in some platforms of metro systems. Figure 9.1 shows a typical PSD installation. The purpose of the PSD function is to control and protect the passenger exchange at the platforms. The PSD function supervises the on-board train doors and, if present, the wayside platform screen doors. It synchronizes the opening and closing of platform doors and train doors. To guarantee passenger safety, the following protective mechanisms are part of the PSD function:

- The train doors as well as the PSDs are only allowed to be opened if the train is stationary.
- The train doors as well as the PSDs are only allowed to be opened on the correct side.
- The PSDs are only allowed to be opened when there is a train in the correct position at the platform (the train doors have to match the related PSDs).
- Only PSD sections that match the train length are allowed to be opened.
- During passenger transfer (open doors) the train must remain stationary.
- If a platform screen door is open unintentionally, no train is allowed to approach the platform.
- If there is a malfunction of the train doors, the PSDs must not open.

9.1.3 Study Design and Data Collection

As an input for the study, Siemens provided the following documents: A glossary describes the terms of the application domain. A system requirements specification (SYS_RS) defines requirements for the TGMT system. In the system architecture spec-

ification (SYS_AS), requirements are given on subsystem level. These documents were taken directly from the PSD development.

The MIRA approach was applied to these documents to specify the requirements specification of the PSD and to perform the quality assurance of the functional requirements. A functional architecture was developed by modeling the functional behavior of the system as a component architecture. This part of the project was conducted over a time period of six months by four researchers including the author. The total effort spent on modeling the requirements specification and the functional architecture in this time period was approximately six person months in total.

The data obtained in the case study was:

1. The resulting *models of the TGMT specification*, including a model of the requirements specification and a model of the functional architecture.
2. The *study results* discussed in the workshops in terms of findings and insights of the project and documented as (non-public) slides and reports.

9.1.4 Project Execution

At the start of the project, two fundamental decisions on the execution of the project were made. Firstly, the modeling should be performed by the research partners that were already experts in the methods, the modeling language and the AutoFOCUS3 tool used in this project. The project was accompanied by workshops and regular phone conferences, where the industrial partners provided the necessary domain knowledge. Secondly, the approach and the data provided in the input documents should remain unchanged for the project. Neither the approach (in this part of the project MIRA) nor the input documents should be adapted to fit each other. The requirements provided by the original input documents have not been adapted to MIRA; AutoFOCUS3 and the MIRA artifact reference structure and tool have not been adapted during the project in order to cope with these documents. However, the MIRA guideline was refined in the course of the project. The study results, models and findings, have been presented to and discussed with experts from Siemens in three full-day workshops.

9.1.5 Data Analysis Process

By applying the MIRA guideline, the input documents from the TGMT were transformed into a requirements specification according to the MIRA artifact reference structure. The researchers elaborated different parts of the requirements specification independently. It was agreed beforehand for each step which researcher would work on which part of the model. If a modeling step led to different results for different researchers, the cause was investigated. This led to an improvement of the MIRA guideline. Findings that positively influenced the quality of the requirements specification were documented, indicating the effectiveness of the approach. For an assessment of the applicability, the resulting models in AutoFOCUS3 and the findings were discussed with the domain experts.

9.1.6 Instantiation of the MIRA Approach

To develop the requirement model, the following steps as proposed by the MIRA approach were conducted. Figure 9.2 gives an overview of the approach.

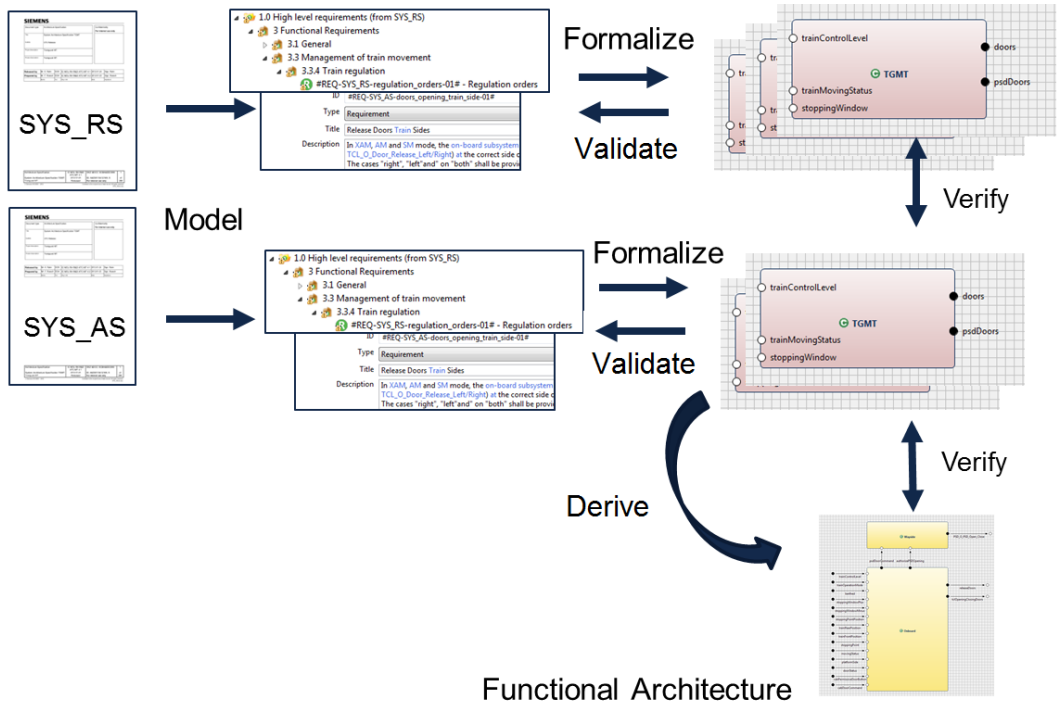

Figure 9.2: The activities performed in the case study on the original documents provided by Siemens follow the MIRA guideline

Elaborate the Requirements Specification in Prose. In the first step, we transferred the informal information from the SYS_RS document and the SYS_AS document to AutoFOCUS3. This information included requirements as well as trace links between requirements and cross references from requirements to their source. The information was structured and classified according to the RE concepts provided by the MIRA approach. The attributes for each RE concept were filled as far as information was provided by the input documents.

We documented the input documents, the SYS_RS document and the SYS_AS document, as `documents` in the MIRA tool. On the system level, we documented systems that interact with the TGMT as `external systems`. On the subsystem level, we documented system functions that interact with the PSD function as `external systems`.

We analyzed the SYS_RS and SYS_AS documents to identify all `functional requirements` related to the PSD function. We checked each of these requirements whether we could classify it as a `scenario`, an `interface requirement` or a `user function` according to the definitions provided by the MIRA approach.

From the SYS_RS and SYS_AS documents, we transferred all `functional requirements` in scope and their *description* unchanged as prose to the MIRA tool.

We added the requirements management information such as *ID*, *title*, and the *author*. An example is given in Figure 9.3. We documented the source `document` as the *requirement source* for each `functional requirement`.

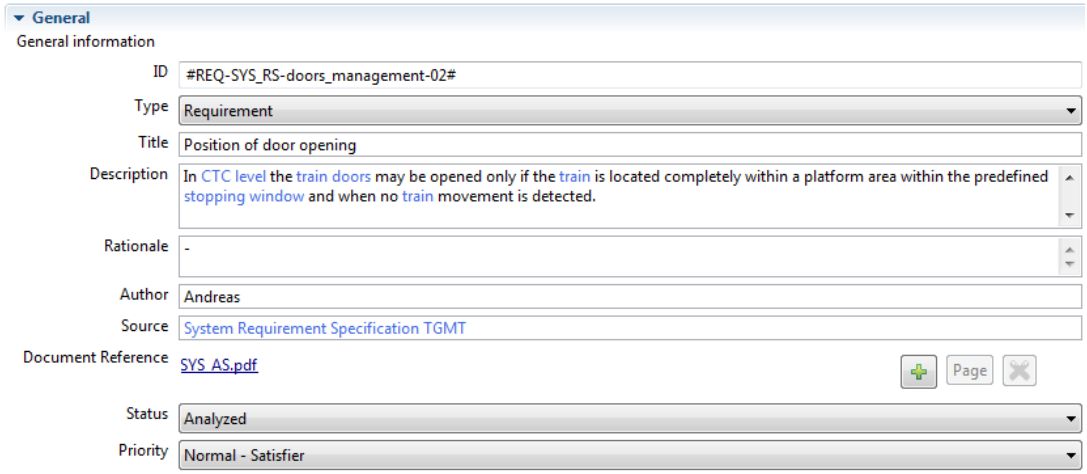

Figure 9.3: Textual description of a requirement

The input documents contained information about the refinement from system to subsystem requirements. These were modeled as `refinement links`, see Figure 9.4. Furthermore, the documents contained so called cross-references between requirements. These were documented as unclassified `trace links`.

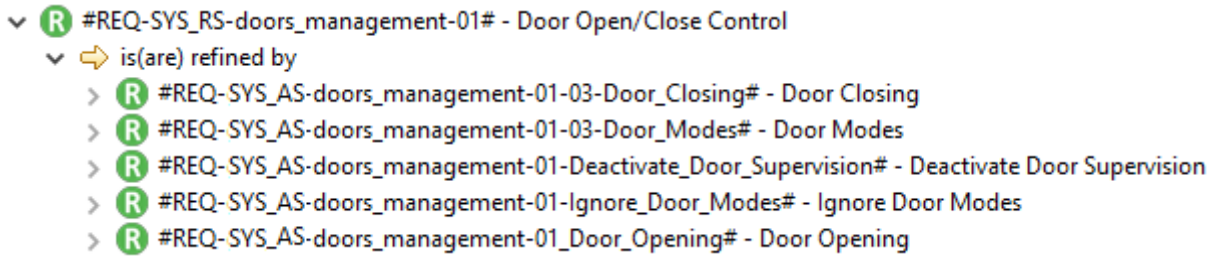

Figure 9.4: Tree view on the refinement links

Designate Terms. For each domain-specific term used in the requirements for the PSD system, a `glossary term` was created. An example is given in Figure 9.5. Whenever available, the *definition* of the term was included from the input glossary. We checked the designation using the term highlighting functionality offered by MIRA.

Formalize Functional Requirements. We analyzed the *description* of each `interface requirement` to identify and document its *design scope*, i.e., the system under development in scope of the requirement. The input documents contained requirements at different levels of abstraction. Requirements from the SYS_RS document had a design scope on the TGMT system. The design scopes of requirements

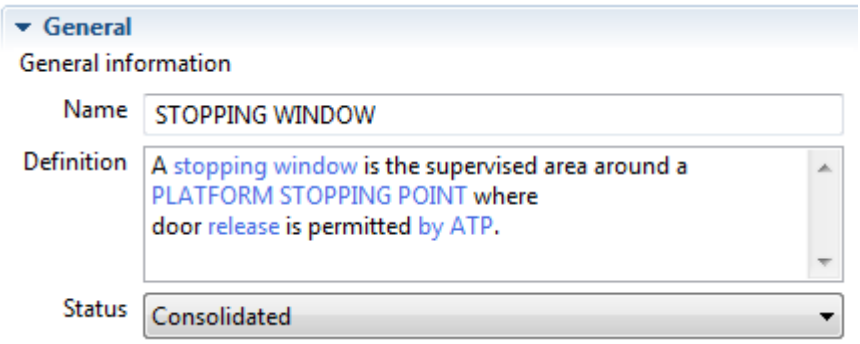

Figure 9.5: Glossary term

contained in the SYS_AS document were the wayside subsystem and the on-board subsystem.

For each `interface requirement`, we analyzed their *description* to identify the *stimuli* and *responses* of the system under development according to the *design scope*. We created corresponding data types for each *stimulus* and *response*. When we identified data types for the same domain concept, i.e., the same `glossary term`, in different requirements, we homogenized them into one data type. Figure 9.6 gives an example for such a data type that stems from two requirements that stated under which conditions the propulsion of the train has to be cut off and reactivated. We then formalized the *stimuli* and *responses* as inputs and outputs of a component in a component diagram. The component represents the relevant system boundaries for this requirement. We specified the required behavior as an I/O assertion. Figure 9.7 provides an example for the formalization of a requirement on propulsion release.

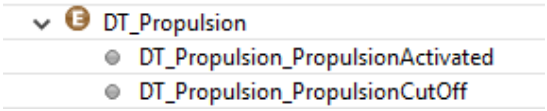

Figure 9.6: Data type that was created based on the interface requirements on propulsion cut off and activation

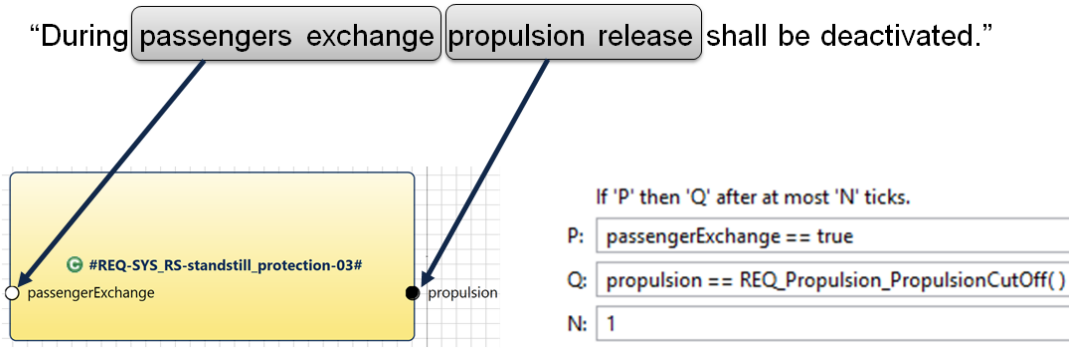

Figure 9.7: Formalization of an interface requirement to a formal interface specification (left) and a formal specification of the interface behavior (right)

We structured the `scenario` according to the scenario template provided by MIRA,

see Figure 9.8. Using the information contained in the scenario template, the `scenario` was formalized to a Message Sequence Chart (MSC), see Figure 9.9.

Scenario

Overview

Name: Door open and supervision scenario in CTC

Success scenario: ☑

Steps

Step	Action	Actor	Action Type	Branch
1	The train arrives at the platform and comes to a standstill within the stopping window. The train doors and the PSDs are all closed.	Train	External	<no branch>
2	The on-board subsystem releases the train doors for opening and sends an PSD open authorisation to the wayside subsystem.	<no actor>	Internal	Door open manual action - Step 1 (The on-board subsystem sends a PSD open command due to a manual driver action.)
3	The on-board subsystem sends a PSD open command due to an automatic door mode control.	<no actor>	Internal	<no branch>
4	The wayside subsystem transfers the PSD open command only to PSDC which fits with the train doors. The other PSDCs at the platform do not receive the open command.	Plattform Screen I	Output	<no branch>
5	The PSDs and the train doors open at the same time.	Plattform Screen I	External	<no branch>
6	As soon as the PSDs open, the wayside subsystem sets the PSD RAUZ to restrictive to inhibit any movement of the train.	<no actor>	Internal	<no branch>
7	The on-board subsystem supervises the train doors during the passenger exchange.	Train	Input	<no branch>

Add | Remove | Move Up | Move Down

Figure 9.8: 'Door open and supervision' scenario structured according to the scenario template provided by MIRA

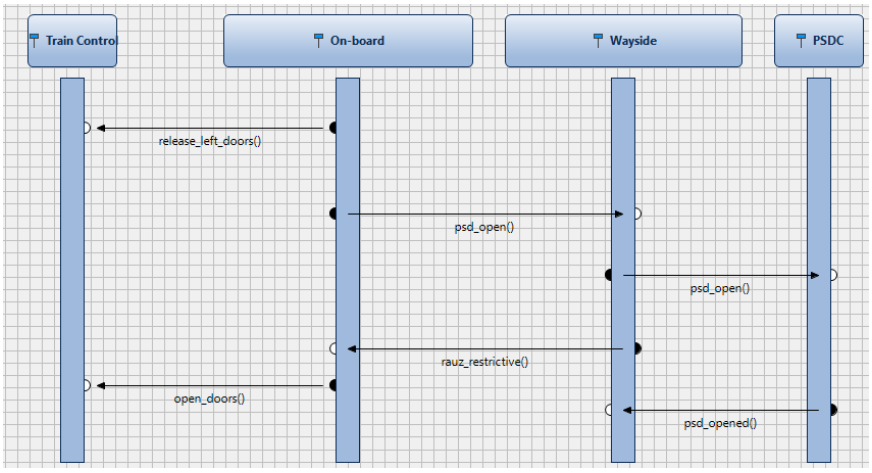

Figure 9.9: 'Door open and supervision' scenario formalized as an MSC

Validate Formal Requirements. We validated the formal representation of the `functional requirements` in a manual review. Goal of the validation was to ensure that the formal representation expressed the required behavior as defined by the textual representation. The check was performed by a different researcher than the one who did the formalization. To obtain confirmation by the domain experts, we presented some of the formalized requirements to them in a workshop.

Analyze the Refinements from System Requirements to Subsystem Requirements. We analyzed the refinements between system requirements and subsystem

requirements based on the `refinement links` to reveal missing refinements, and missing or incomplete system or subsystem requirements.

The documented `refinement links` facilitated automated structural checks on the refinement. We performed the following syntactical checks on the documented refinement links:

- Is every system requirement linked to a subsystem requirement?
- Is every subsystem requirement linked with a system requirement?

In the next step, we set up formal refinement specifications based on the refinement links to further check the syntactic and semantic contents of the refinement. An example for such a formal refinement specification is sketched in Figure 9.10.

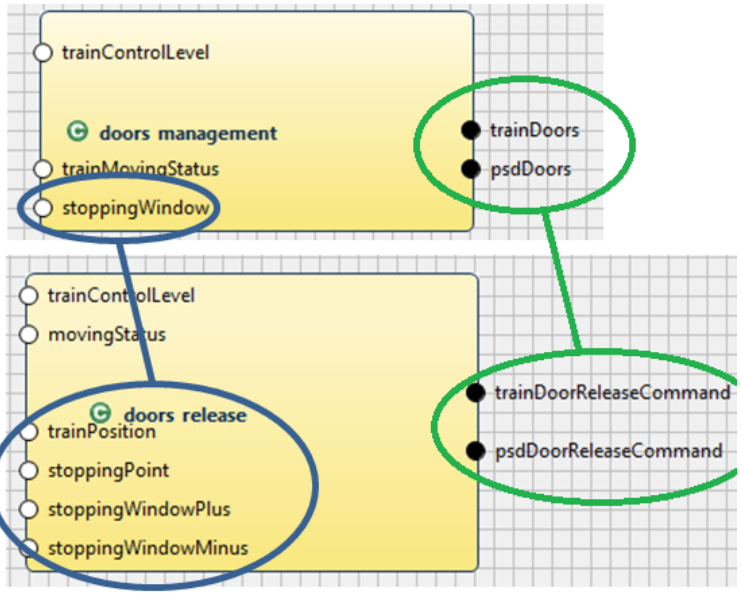

Figure 9.10: A formal refinement specification defines the mapping of input ports (left) and the output ports (right) of the refinement from the requirement on door management to the requirement on door release

We performed the following checks on the formal refinement specifications:

- Is each *stimulus* or *response* (modeled as inputs or outputs) of a system requirement refined to a *stimulus* or *response* of a subsystem requirement?
- Does each *stimulus* or *response* (modeled as inputs or outputs) of a subsystem requirement originate from a *stimulus* or *response* of a system requirement? If not, could it origin from another system requirement?

Synthesize Interface Requirements into User Functions. We manually synthesized the `interface requirements` for the onboard and wayside subsystems to `user functions`. Therefore, we identified all `interface requirements` that described the same *response*. For these requirements, we created a `user function` that realizes all requirements. Each `user function` was represented as an executable model, for example, a state automaton or a code snippet. We synthesized the formal representations of the `interface requirements` to obtain this executable

model. We used these `user functions` as the input to construct the functional architecture of the PSD function.

For example, the case study contained two `interface requirements` on the propulsion release of the train, a safety mechanism that avoids that the train can move as long as the train and PSD doors are allowed to be opened. The first requirement was formalized to two I/O assertions, the second requirement was formalized to one I/O assertion. These two requirements were synthesized into the `user function` 'propulsion function'. Figure 9.11 shows the syntactic interface of the `interface requirements` that we synthesized into the syntactic interface of the 'propulsion function'. In the functional viewpoint, the `user functions`, respectively their executable models, were composed into a coherent model of the functional architecture of the system, see also B'ohm et al. [BJV+14].

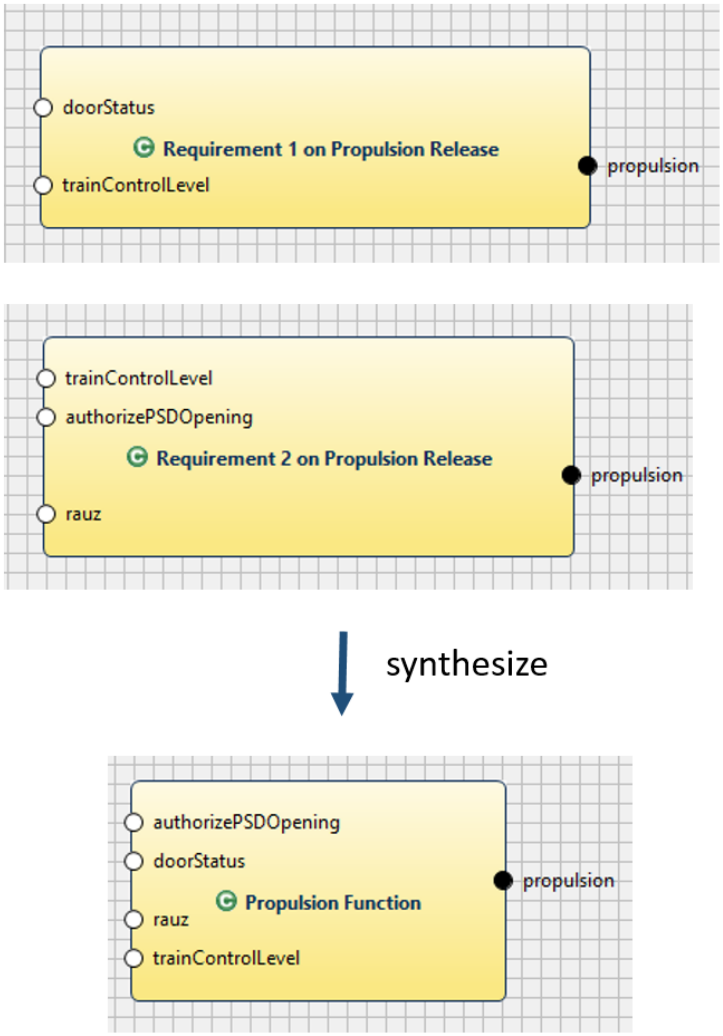

Figure 9.11: The syntactical interface of the user function 'propulsion function' is synthesized from the syntactical interfaces of the two requirements on propulsion release

Analyze User Functions. We analyzed the formal representation of user functions that were modeled as state automata on non-determinism and unreachable states.

Document and Formalize Realization Links. We linked each `interface requirement` by a `realization link` to the components of the functional architecture that realize the required interface behavior. An example for a `realization link` is given in Figure 9.12.

We specified these `realization links` as a formal refinement specification which defines the mapping between the input and output ports of the component diagram that represents the syntactic interface of the `interface requirement` and the component that realizes this requirement. Figure 9.13 shows a simple example of such a mapping. Here an input port of the formal requirement (denoted as source) can be mapped directly to an input port of the component of the functional architecture (denoted as target) that satisfies this requirement.

Verify User Functions against Interface Requirements. The formal refinement specification facilitated an automated transformation of the I/O assertions to the user functions. Thereby, the model of each `user function` could be verified against the I/O assertions of the `interface requirements` using formal verification. If the formal verification failed, it provided a counter example, that we simulated to investigate how the I/O assertion was violated.

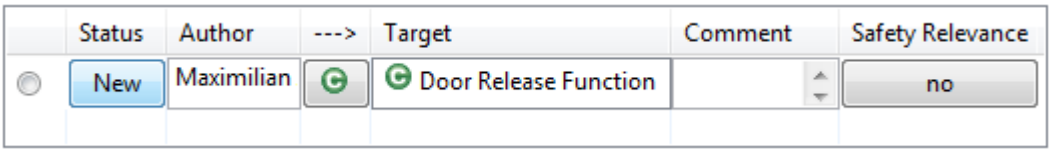

Figure 9.12: Specification of a realization link from a requirement to the door release function

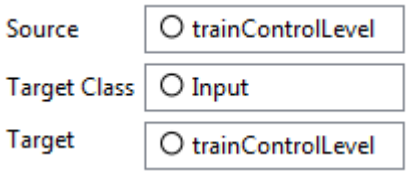

Figure 9.13: Specification of a formal refinement specification

Validate the User Functions. Both the researchers and the domain experts simulated the user functions and their composition in the functional architecture to validate the required system behavior.

Verify the Functional Architecture against the Scenario. We applied the MSC Conformity Check provided by the AutoFOCUS3 tool to investigate whether the 'Door open and supervision' `scenario` in its formal representation as an MSC can be executed in the model of the functional architecture of the system.

9.1.7 Study Results

Coverage of the MIRA Approach. The contents of the input documents were classified as `glossary terms`, `documents`, `external systems`, a `scenario`, `interface requirements`, `refinement links` and unclassified `trace links`. The interface requirements were manually synthesized into `user functions`. We created `realization links` between the `functional requirements` and the functional architecture. During the quality assurance activities, we documented the results in the form of `conflict links` between requirements and in `QA check lists`. The case study did not instantiate `goals`. We applied model-based techniques for the analysis, validation and verification, but we did not generate and execute test cases on the RE level (but in subsequent development steps).

Elaborate the Requirements Specification in Prose. From the input documents, we extracted in total 29 requirements which were in scope of the study. One requirement was classified as a `scenario`. 25 requirements were classified as `interface requirements`. These requirements contained required interfaces and interface behavior of the PSD as well as required communication with the environment of the PSD. Three requirements remained unclassified, as they did not did prescribe behavior.

Designate Terms. All researchers working on the requirements specification considered the designation of `glossary terms` as a useful support. The quick access to the glossary improved the understanding of the domain. As the academic partners were not experts in the domain of rail automation, the lack of domain knowledge of important concepts could to a certain extent be mitigated by the integrated glossary.

The designation of terms revealed undocumented domain knowledge as some term definitions were missing in the input documents.

Formalize Functional Requirements. We formalized the `scenario` and the 25 `interface requirements`. The `scenario` was formalized to an MSC. Each of the `interface requirements` was formalized to one to five I/O assertions. The reason for this is that some of the requirements contained more than one statement on required interface behavior or more than one I/O assertion was necessary to cover a statement. Six `interface requirements` were only partly formalized, as they contained real-time behavior that was out of scope for this project, included information other than interface behavior or described behavior that was too abstract or vague to be formalized. Table 9.1 gives an overview of the formalization and Table 9.2 gives examples of information that could not be formalized.

Two observations in the course of the formalization process showed the need for a strict guidance of the formalization process and subsequently led to improvements of the MIRA approach.

The first observation was that without a strict guideline on the formalization, every person that formalized requirements applied a different modeling notation (MSC,

Table 9.1: Numbers from the Case Study

Input document	Requirements in scope	Classified Requirements	Formalized	Thereof only partly formalized
Sys_RS	7	4	4	2
Sys_AS	22	22	22	4

Table 9.2: Reasons and examples of pieces of information that could not be formalized

Reason	Requirements in scope
Too vague	“The TGMT system shall support run authorization zones.”
No interface behavior	"The train position allowing the door opening and the subset of doors allowed to open have to be defined by engineering."
Real time	“The on-board subsystem shall use TP DOOR COMMAND DELAY to synchronize the control of the train doors and the PSDs.”

I/O assertion, state automaton) and thereby formalized different aspects of the requirements. Therefore, the MIRA guideline now recommends concrete modeling notations for functional requirements.

Our second observation concerned the number of requirements we could formalize. We performed the formalization in two iterations. In the first iteration, we could only formalize 21 system and subsystem requirements. The second iteration yielded 26 formalized requirements, an increase of 24%. The increase could be achieved by explicitly defining to extract the *design scope*, *stimuli*, *responses* and *interface behavior* from the textual requirements and by defining explicit mapping rules from this information to the formal representation. A literature search revealed that several model-based RE approaches recommend controlled natural language as an intermediate step in the formalization to extract information before formalizing it. As a consequence, the MIRA artifact reference structure was extended for `interface requirements` in order to be able to document the results of this structuring activity.

The formalization of requirements revealed an ambiguity in a requirement; a phrase could be interpreted in two ways and the correct meaning had to be clarified in a workshop.

Validate Formal Requirements. The validation confirmed that the formal requirements reflect the textual requirements.

Analyze the Refinements from System Requirements to Subsystem Requirements. Checks on the `refinement links` did not reveal any quality issue. For more detailed investigations, we specified formal refinement specifications of the `refinement links`. The case study revealed that setting up the formal refinement links between requirements on the two levels of abstraction, the system and subsystem level, was a difficult task and could not be performed in the first place. In Section 9.1.7, we discuss our results of analyzing the underlying problem and the

solution that we applied to overcome this difficulty.

Setting up and analyzing the formal refinement specifications identified three quality issues. A refinement from system to subsystem requirements was incomplete. *Stimuli* defined on the system level were not documented explicitly on the subsystem level. We stated the stimuli explicitly in the subsystem requirements. In another case, a *stimulus* at the subsystem level did not originate from any system requirement. We added a system requirement in order to complete the refinement. In a third case, a refinement from a system requirement to a subsystem requirement was missing and we added a `refinement link`.

We performed the refinement analyses manually, as the MIRA tool did not offer automated checks. Some of these checks could be automated, for example, using database queries or model-checking.

Gap between System and Subsystem Requirements. One of the main results of the case study was that the formal refinement specifications between the given system and subsystem requirements may become very complex. A detailed discussion of this finding and our solution is provided in [TBP14]. In the following, we provide a summary of our investigations.

An analysis of the refinement revealed that the complexity results from the fact that two refinement operations, namely *interface refinement* and *decomposition*, were performed in one step.

By *interface refinement*, we understand a refinement of requirements on the system interface (without adding architectural decisions). For example, the system requirement defines an abstract *stopping window* in which the train has to stop. The subsystem requirement defines detailed interval boundaries for the stopping window.

By *decomposition* of system requirements to subsystems, we refer to the break-down of requirements to subsystems (without changing the system interface) that results in a change of the design scope of a requirement. For example, the system requirement *doors management* defines an abstract stopping window within which the train has to stop. The subsystem requirement *doors release* details the stopping window to a stopping point for the train endpoints with tolerances, while at the same time assigning the calculation of the stopping window to the on-board subsystem.

We call the inclusion of both refinement operations in one refinement step a "gap" between system and subsystem requirements. There are several consequences of such a gap:

Specification: In the case of a gap defining the formal refinement specification is a challenging task: The information of all other subsystem requirements belonging to one decomposition has to be included in the formal refinement specification and to be integrated with the interface refinement. This results in complex representation and interpretation functions and is a possible source of errors.

Quality assurance: If there is no formal refinement specification, as a consequence formal analysis and verification cannot be performed automatically. Decomposing requirements may lead to a scattering of system requirements (i.e. system behavior) over several subsystem requirements. In this case, analyzing requirements regard-

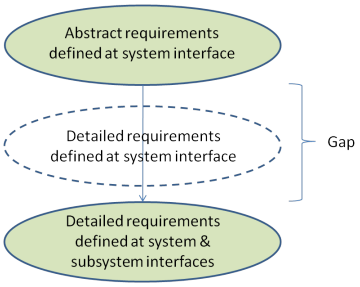

Figure 9.14: Detailed system requirements close the gap between system and subsystem requirements

ing their interface refinement is more challenging, as it is necessary to identify and consider all relevant subsystem requirements for the analysis.

For example, the validation of the interface refinement against the expectations of stakeholders is complex, as the stakeholders also have to incorporate the decomposition. Manual checks on whether the behavior of subsystem requirements conforms to the behavior required in the refined system requirement are challenging as the decomposition has to be considered.

Reuse: Documenting system interface refinement only on subsystem level also hinders reuse of the refinement information when the architecture changes. In contrast, if the interface refinement is already documented on the system level, then reuse of concretization information is possible even when the decomposition of the system into subsystems changes.

In order to close the gap, we introduced a new level of requirements called *detailed system requirements* that have a design scope on the system, but contain the interface refinements of the subsystem requirements. The detailed system requirements contain all interface refinement information of the subsystem requirements that are visible at system level. These detailed system requirements connect the abstract system requirements with the detailed subsystem requirements. Figure 9.14 depicts the refinement from system requirements to subsystem requirements. We introduced a bottom-up restructuring approach that provides transformation operations for subsystem requirements in order to obtain the detailed system requirements. The restructuring approach resulted in 14 detailed system requirements.

The restructuring approach required additional specification effort as we had to specify more requirements and more refinement links. This also increased the overall number of formal requirements and formal refinement specifications. Nonetheless, the introduction of the detailed system requirements could overcome the problems related to the gaps:

Specification: By separating interface refinement and decomposition, information on subsystems is not integrated with interface refinement when specifying representation and interpretation functions. Therefore, these functions become easier to specify. This eliminates a possible source of specification errors.

Quality assurance: Quality assurance of the interface refinement, such as the validation against the expectations of the stakeholders, or the verification of requirements against the implementation, can now be performed on system level independent from the system decomposition.

Reuse: Detailed knowledge of the system interface is now available not only at subsystem level, but also on system level. This shift of knowledge from subsystem to system level enables reuse of the interface refinement information when changing the decomposition of the system into subsystems. The new requirements layer facilitates the understanding of the rationale behind decisions by separating interface refinement and decomposition. An additional benefit is the possibility to introduce exception/fault scenarios on system level based on the detailed system requirements.

From our point of view this justifies the additional specification effort. The investigation also revealed an open research question. In our case study, some of the detailed system requirements were composed of numerous subsystems requirements. This led to bloated requirements, in terms of extensive input and output port definitions. These requirements also summarized several statements on the interface behavior of the system. The size of these requirements made it labor extensive to validate that the formalization of each requirement conformed to the textual specification and to analyze the formal refinement specifications regarding completeness. This points to the open research question regarding the size of requirements that is best suited for the various activities performed on them.

Synthesize Interface Requirements into User Functions. The 22 `interface requirements` on the subsystem level were synthesized into nine `user functions`, thereof five for the onboard system, and four for the wayside system. We composed these user functions into a model of the functional architecture of the PSD function.

The synthesis revealed that two requirements contained conflicting conditions on the same system response. A discussion with the domain experts revealed that both requirements had to be synthesized into one requirement to overcome this conflict.

An interesting finding in developing the functional architecture was that the *stimuli* of some requirements were *responses* defined in other requirements, leading to functional dependencies between requirements. We called these dependencies "modes". Modes summarize conditions on sequences or sequence histories of *stimuli* that affect the system behavior. Defining dedicated modes for these conditions avoids to specify them redundantly in the requirements. For example, a requirement defined the conditions, under which the 'door open command' is stated. Another requirement referred to this 'door open command'. Without this mode, all condition would have to be defined repeatedly in the second requirement. Modes are not visible at the system boundary. Thereby, the notion of modes challenges the idea of pure black-box requirements on the system interfaces. Vogelsang [Vog15] provides a deep investigation of the occurrences of these functional dependencies and their consequences in practice, suggesting solutions on how to approach them.

Analyze User Functions. The analysis did not identify any quality issues.

Verify User Functions against Interface Requirements. The formal verification of the `user functions` against the subsystem `interface requirements` confirmed that the functional architecture fulfilled 18 of the 22 formalized subsystem re-

quirements. Five requirements could not be verified because of tool issues. For three requirements, the verification failed. A failed verification points to possible quality issues that needs to be discussed with the stakeholders. The researchers investigated the failed verifications and revealed that requirements were conflicting. Discussions with the domain experts revealed that the requirements were conflicting, because certain properties in the context of the PSD function were not explicitly documented in the requirements. Furthermore, the verification revealed that we made a mistake in the model of a `user function`, which we then corrected.

Validate the User Functions. The simulation of the `user functions` revealed that an `interface requirement` was missing. This missing requirement was related to a condition for closing the train doors before train departure. After adding this requirement, the PSD function reacted as expected.

Verify the Functional Architecture against the Scenario. The MSC Conformity Check proved that the 'Door open and supervision' `scenario` is feasible in the functional architecture of the system.

9.1.8 Benefits of the MIRA Approach

Despite the high quality of the input documents, the application of the MIRA approach resulted in concrete findings that could be used to improve the input documents. The application of the MIRA approach yielded several findings in the input documents during the designation of terms, formalization, analysis, validation and verification of requirements. The quality improvements obtained by applying the MIRA approach are summarized in Table 9.3. Ambiguities, inconsistencies and incomplete requirements were identified that prompted discussions with the domain experts. The discussions revealed that the main reason for these quality issues was undocumented domain knowledge. The case study confirmed some of the results of the study on model-based quality assurance in RE that was presented in Chapter 4. The case study even provided new insights that were not identified in Chapter 4. The case study provided evidence that a synthesis of formal interface requirements into a formal user functions can detect inconsistencies and that specifying a formal refinement specification can identify missing requirements and trace links, thereby increasing completeness.

The MIRA approach revealed a gap between system requirements and subsystem requirements that could be analyzed based on the underlying formal system modeling theory. The suggested approach for closing this gap can result in a less error-prone specification, has a positive impact on the verification and can ease the reuse of requirements when the system architecture changes.

The MIRA approach was also beneficial for subsequent development activities. Most of the effort in the project was spent on requirements modeling. This included the creation of the glossary and the formalization of requirements. However, this effort facilitated other activities. The formalization of requirements greatly sped up the creation of the functional architecture, as its structure could be extracted directly from the requirements. The functional architecture could, to a large degree, be derived

Table 9.3: Quality improvements achieved in the case study by applying the MIRA approach

Quality Improvements
Adequacy
The MIRA approach increased the adequacy of system and subsystem requirements by identifying missing refinement links between system and subsystem requirements and a missing system requirement.
The MIRA approach increased the adequacy of user functions to their functional requirements by identifying a fault made by a researcher in the model of a user function.
Unambiguity
The MIRA approach decreased lexical ambiguity of the terms used in functional requirements by identifying domain terms without description.
The MIRA approach removed a syntactical ambiguity caused by alternative interpretations of the description of a functional requirement.
Completeness
The MIRA approach increased the logical completeness of functional requirements by identifying a missing requirement for closing the train doors.
Consistency
The MIRA approach increased the logical consistency of functional requirements by identifying requirements with conflicting conditions on the same system response.
The MIRA approach increased the logical consistency of functional requirements by identifying conflicting requirements because of undocumented properties of the context of the PSD function.

from the formalized requirements in a straightforward manner. The development from a logical architecture to implementation code could be conducted with approximately the same effort as it was needed for the development of a requirements specification and the functional architecture. During these phases, no quality issues were detected that originated from functional requirements or an inadequate system behavior.

9.1.9 Threats to Validity and Limitations

The case study was faced with the following threats to validity and limitations.

Quality Assurance. The study object is an industrial specification of a system that had already been implemented. This might be a threat to *internal validity*, as the input specifications already had a very high quality and only minor issues could be detected during the quality assurance. This high quality of the input specification also influenced the classification and formalization of requirements, as this step might have been more difficult with an input specification with lower quality.

Application by Researchers. The application of the MIRA approach has been conducted by a group of researchers with a profound knowledge of the MIRA approach, the AutoFOCUS3 tool and the modeling language. This had an effect on the results with respect to the applicability of the MIRA approach. It represents a threat to the *external validity* of the case study, when results should be generalized to domain experts. The researchers demonstrated and discussed the application of the MIRA approach to the domain experts in regular workshops. Nonetheless, the case study is not designed to demonstrate the applicability by domain experts. The *reliability* of the case study can be influenced by this decision, as it is possible that other persons might interpret the MIRA guidelines differently.

Time Estimates. Realistic effort estimations and time consumptions cannot be deduced from this case study. The study was not performed in the context of the Siemens organization. The time frame did not correspond to a realistic environment with tight deadlines. Furthermore, the time period for conducting the project included the time required for modeling, but additionally also the time to research, present and discuss the study findings and document the case study results. A way to investigate usability and realistic time estimates would be to train experts from industry in the MIRA approach and tool and then perform the case study by the trained domain experts in a realistic environment.

Domain. The case study applied the MIRA approach to a study object from the train automation domain. The generalizability with respect to other domains cannot be guaranteed by this study.

Other case studies with different settings have applied MIRA in the context of embedded systems (see Section 9.3). The generalizability of the sum of these studies is discussed below.

Usability. The modeling was undertaken by the academic partners, who were experts in the MIRA approach, the tool and the modeling language. Therefore, the usability of the MIRA approach could not be assessed in the case study.

9.1.10 Summary

The case study showed that the MIRA approach was applicable for this study object. We could model the PSD by applying the MIRA approach. The MIRA approach was applied on two levels of abstraction, the system level and the subsystem level. We validated the textual and formal requirements with the Siemens project members in the workshops. We analyzed the requirements and their refinement. The formal requirements and the formal refinement specifications were used to verify subsystem requirements against system requirements and the functional architecture against subsystem requirements. The application of MIRA improved the quality of the Siemens documents. Furthermore, the application refined the MIRA approach and resulted in an improved and more detailed guideline and attributes for the description of interface requirements.

9.2 Case Study on Extensibility

The Liebherr Case Study was conducted within the SPES-XT project[2]. The study object was a flight control system. The purpose of the collaboration between Liebherr and fortiss was to develop methods that facilitate the synthesis of deployments for flight control systems. In particular, the automatic deployment generation for Liebherr flight controls was investigated. This deployment must fulfill various requirements with respect to safety, communication, storage and timing. For modeling and deployment generation, AutoFOCUS3 was used.

9.2.1 Study Object

Modern aircraft must satisfy strict requirements on safety, performance and security. New functionality in their software often requires additional hardware. Any increase in hardware must be accommodated within limited space. Furthermore, it increases production costs, aircraft weight and power requirements. To address these challenges, modern aircraft architectures include complex embedded cyber-physical systems. An example is the flight control system, which is responsible for the movable surfaces that control the aircraft's direction in flight. The study object is a flight control system developed by Liebherr. For this case study, Liebherr provided a set of requirements that constrain the deployment of the flight control system. Based on these requirements, deployments had to be synthesized for different configurations of software applications and execution platforms.

9.2.2 Research Objectives

The goal of the part of the case study presented here was to show the extensibility of the MIRA artifact reference structure and tool. The Liebherr Case Study extended the MIRA artifact reference structure and its implementation in the MIRA tool to be able to specify domain-specific requirements and to trace these requirements to subsequent component architectures. In order to document the respective requirements, MIRA was extended with a set of templates for project-specific requirement types. The extensibility of the MIRA artifact reference structure and tool regarding new requirement types was evaluated. This case study is presented more briefly compared to the first case study, as it only demonstrates this specific aspect of MIRA.

9.2.3 Study Design

The case study was conducted using the tool AutoFOCUS3. The MIRA tool is implemented as a plug-in for AutoFOCUS3. Accordingly, the requirement types were implemented and the requirements were modeled using AutoFOCUS3. A staff member of the same research group as the author conducted the implementation of the extensions.

[2] http://spes2020.informatik.tu-muenchen.de/spes_xt-home.html

9.2.4 Results

MIRA Extensions. In the scope of the Liebherr case study, four requirement types were distinguished; communication, storage, safety and timing. The MIRA artifact reference structure (implemented in the MIRA tool as an EMF model) for the new requirement types was developed by inheriting from the generic `requirement`. Figure 9.15 shows the definition of the Liebherr requirements. The new requirement types were annotated with specific attributes, for example, a communication requirement should contain the communication type and the amount of bandwidth needed. No additional changes on the artifact reference structure were necessary.

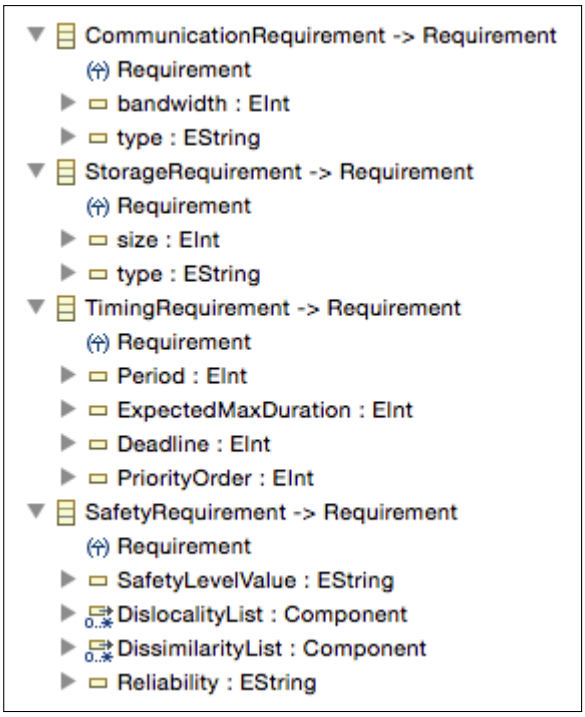

Figure 9.15: EMF models for the Liebherr requirements

GUI Extensions. The new requirement types were included in the GUI of AutoFOCUS3 by defining those elements which were required by the AutoFOCUS3 modeling framework. For the GUI representation of requirements only the detail section of the existing requirements template had to be extended with the specific Liebherr attributes. Figure 9.16 shows the requirement menu that includes the Liebherr requirements.

Extension of MIRA Operations. No further extensions on the MIRA operations were necessary.

Size of the Case Study. Each software component of the Liebherr study object must satisfy several requirements and often more than one requirement of a particular type. Figure 9.17 shows the requirements created for a software application. This component claims bandwidth on three distinct interconnects and also requires three

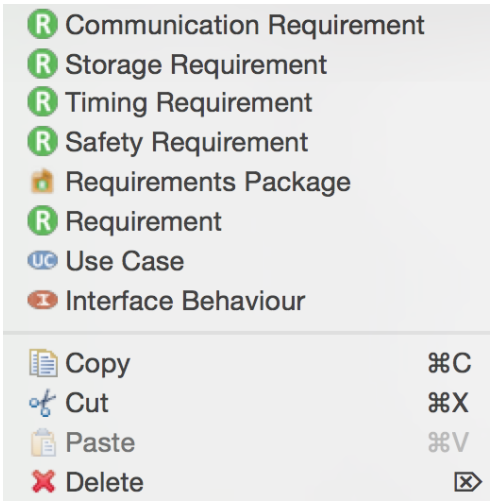

Figure 9.16: Requirement menu

separate blocks of memory. In the Liebherr model, a component is always associated with at least seven to eight requirements. Based on these requirements, deployments could be synthesized for several configurations of software applications and execution platforms. The largest configuration contained 25 software applications with 190 requirements and 25 execution platforms.

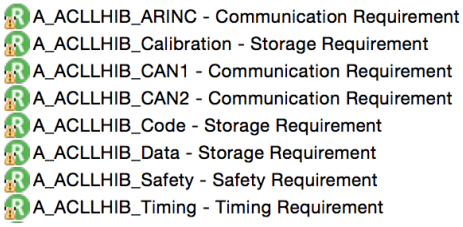

Figure 9.17: Requirements of a control loop software application

Perceived Difficulty of the Extension. The work needed to extend the requirement framework was simple and intuitive, mostly driven by the AutoFOCUS3 development conventions. It was sufficient to extend the requirements data model and the GUI as described above. No new functions needed to be implemented in AutoFOCUS3 for the integration of the new requirement types. Moreover, for this case study the existing tracing links between requirements and components could be used. The case study could benefit from the fact that requirements of the same type could be modeled easily, which was an important feature that the MIRA framework offered.

9.2.5 Threats to Validity and Limitations

Experience with the MIRA Approach. The researcher was familiar with the MIRA approach, but not particularly experienced in the MIRA approach, in AutoFOCUS3 development or involved in the previous development activities of MIRA.

Implementation Support. During the implementation, the researcher was supported by the developers of the MIRA tool. The collaboration helped the researcher to identify the extension points more quickly. This may influence the perceived facility of the extension.

Domain. The researcher evaluated the extensibility in one case study in the flight control system domain. In order to generalize the results, several case studies in different domains would need to be conducted.

Coverage of the MIRA Approach. The case study reflects the extensibility of new kinds of requirement types by adding additional templates for these types. No additional formal models or new kinds of operation on the artifact reference structure were added. The case study did not formalize requirements to a formal representation. In order to generalize the results, several case studies would need to be conducted that extend different aspects of MIRA.

9.2.6 Summary

The Liebherr Case Study showed that an extension of MIRA with new requirements types is possible within few steps. The use of the MIRA tool facilitated the development and specification of domain-specific templates for requirements (communication, storage, safety, timing) as well as tracing them to subsequent component architectures.

9.3 Further Studies with MIRA

Researchers extended the MIRA approach with respect to safety information.

Safety Attributes for Requirements. A group of researchers extended MIRA with capabilities to specify safety-relevant information on the requirements level. Annotating a safety-level for each requirement provides the capability to track and trace this information in the system development and use these requirements as an input for safety analysis. How to integrate a safety analysis in MIRA and its implementation in AutoFOCUS3 is for example described in [VSKC13].

Researchers applied MIRA in the following case studies:

Pacemaker Software. Gareis [Gar12] modeled the requirements for a pacemaker software in AutoFOCUS3 using MIRA. Requirements were specified textually and modeled as state automata or I/O assertions. Based on the state automata, test cases were generated and performed on the model of the system architecture. The I/O assertions were verified by applying model-checking. The model of the architecture was tested against a Matlab/Simulink model of a heart.

Monitoring and Control System for a Desalination Plant. Campetelli et al. [CJB+15] applied MIRA to model parts of a desalination plant. The study object was a monitoring and control system that supervises the water level of the seawater tank of the desalination plant. MIRA was applied to specify the textual and formal functional requirements. The formal requirements were verified at an early stage in the development process. The authors reported that the tool support was suitable. The authors reported benefits of the formalization of requirements: Obtaining more precise requirements with defined semantics and avoiding errors and incomplete requirements. The case study furthermore confirmed the conformance of MIRA with the SPES development method.

Canal Monitoring and Control System. AutoFOCUS3 together with the MIRA tool has been used in a series of master-level practical courses on model-based engineering for students at the Technische Universität München. Vogelsang et al. [VEH+14] report on the results of two consecutive courses. They describe the set of artifacts and the concrete process that was applied in these two courses. The case that was modeled in these two courses is a canal monitoring and control system. The courses developed a system model from requirements to an architecture model from which code could be generated. The students worked in two groups: The first group developed the requirements specification and the second group developed the system architecture. MIRA enabled the continuous conformance assessment between requirements and architecture, leading to a high confidence that requirements and architecture conform to each other. The requirements were specified textually based on an input specification. The case study applied the use case and scenario templates provided by MIRA. Scenarios were formalized by MSCs. A set of rules was provided to check the formalization results. MSCs were finally used to automatically check the resulting architecture model for conformance with the scenarios.

9.4 Conclusion

The case studies presented in this chapter indicate that the MIRA approach is effectively applicable in different domains and by different user groups.

User Groups of MIRA. The case studies reported in this chapter were either conducted by researchers in collaboration with the author, by researchers independently of the author, or by students guided by other researchers.

Quality Assurance. The application of MIRA in case studies yielded concrete findings in the resulting requirement models that increased the quality of the requirement models, especially the functional requirements. The MIRA approach scaled to the size of several user-visible system functions in industry and to the size of academic examples of monitoring and control systems.

Domains. Case studies with MIRA were conducted on a set of cases from industry and academia. These cases covered the domains train automation system, pacemaker

software and a desalination plant, thereby covering some parts of the embedded systems domain.

In a second type of projects, researchers provided extensions of MIRA with further requirement types and attributes, demonstrating its extensibility.

Nevertheless, some aspects of the MIRA approach remain subject to further investigations. We did not investigate the application of MIRA by practitioners in realistic circumstances. This could provide insights into costs and benefits of the MIRA approach. A systematic investigation of the scalability of the MIRA approach could alleviate the application of MIRA in industry. As MIRA includes a variety of manual activities, these aspects are highly intertwined with research on the usability of model-based RE approaches. In the investigation of usability, engineering aspects are influenced by psychological aspects and therefore require a carefully designed research strategy that goes beyond the research objective of this work.

Chapter 10

Summary and Outlook

This chapter summarizes the contributions of this thesis and provides possible directions for further research.

10.1 Summary

This thesis is based on the research question "How to specify functional requirements in a seamless model-based development approach for embedded software and systems engineering in order to enable the model-based analysis, validation and verification for both textual and formal representations?' (see Section 1.3). This thesis systematically investigates this problem, presents a suitable solution, and validates and evaluates this solution.

10.1.1 Challenges

In this thesis, we suggest the MIRA approach for the model-based specification and quality assurance of functional requirements. The MIRA approach provides a solution for the four challenges listed in Section 1.2:

Challenge 1: Supporting the Textual and Formal Representation of Requirements. The MIRA approach supports heterogenous requirements specifications where a user can specify functional requirements as informal prose or using a semi-formal or formal representation. Furthermore, MIRA provides support for the quality assurance of both textual and formal representations.

Challenge 2: Covering the Analysis, Validation and Verification of Requirements. MIRA enables model-based quality assurance of the functional requirements. The MIRA guideline defines specification activities on the artifact reference model that aim at avoiding quality issues constructively. Furthermore, the specification enables a set of analytical model-based techniques to support quality assurance:

- Analysis by database queries and formal techniques,

- Validation by database queries and simulation,
- Verification by database queries and formal techniques.

Challenge 3: Integrated in a Seamless Development Approach. MIRA is integrated in a seamless model-based development approach and thereby enables the model-based verification of functional requirements. The model-based development approach consists of the formal system modeling theory FOCUS and the SPES modeling framework that structures the system under development into viewpoints and abstraction layers. The MIRA artifact reference model defines a reference for the contents of a requirements specification, relates these to FOCUS and integrates the contents into SPES.

Challenge 4: Providing Guidance and Tool Support for Challenges 1 - 3 The MIRA guideline offers method support for the MIRA approach by defining the necessary steps to instantiate a requirements specification according to the MIRA artifact reference model. The MIRA tool is embedded in the model-based software and systems development tool AutoFOCUS3. MIRA uses the model-based specification and quality assurance capabilities of AutoFOCUS3 for requirements engineering.

10.1.2 Contributions

In the following, we present the contributions of this thesis.

10.1.2.1 Systematic Investigation of Model-based RE for Quality Assurance

The first part of this work was a systematic analysis of the characteristics that a model-based RE approach should possess. The investigations determined the essential characteristics that MIRA should meet.

Contribution 1: Quality Factors of a Requirements Specification that Enable Model-based Quality Assurance. The goal of the first study was to identify the means to improve the effectiveness and efficiency of quality assurance of functional requirements through model-based RE. *System quality factor* are those characteristics of an artifact, for example, of an RE concept, that have an effect of the quality of the system under development. Quality factors with a negative effect are the so-called *quality issue*. The system quality factors that we investigated in the study are *completeness*, *consistency*, *unambiguity* and *adequacy*. The corresponding target quality issues are *incompleteness*, *inconsistency*, *ambiguity* and *inadequacy*. The study investigated two means to increase the quality of functional requirements with respect to the system quality factors. Firstly, the quality can be increased constructively during the specification through rules that ensure the absence of specific quality issues. Secondly, the quality can be increased analytically through the analysis, validation and verification of requirements. Improving the efficiency and effectiveness of these quality assurance activities directly affects the quality of the requirements. We call

those characteristics of the requirements specification with an impact on the quality assurance activities *QA quality factors*.

The study investigates how a set of RE concepts and their QA quality factors positively impact a set of system quality factors of the system under development through quality assurance. The contribution of this study is a literature study of evidence for these positive impacts. The study identified a set of RE concepts whose definition can positively impact quality assurance. Furthermore, the study investigated QA quality factors that structure and represent these RE concepts in order to enable analytical model-based quality assurance. Each of the concepts and their QA quality factors lead to a more effective or efficient quality assurance of functional requirements. Thereby, they lead to an improvement in the quality of these requirements with respect to the system quality factors. The results of this study have been incorporated directly in the MIRA approach: RE concepts and QA quality factors in the MIRA artifact reference structure; the activities in the MIRA guideline; impacts on efficiency through automation in the MIRA tool.

Contribution 2: Requirements for a Model-based RE Approach. The contribution of this study is a list of 31 requirements for a model-based RE tool. The requirements result from a systematic literature research and consolidated and amended in a questionnaire amongst RE practitioners. The requirements include the various representation forms of requirements that a tool should support. Furthermore, the requirements include tracing abilities of the tool, interfaces of the tool and quality assurance techniques. The last two categories of requirements, requirements on abstraction layers that the tool should support and requirements on guidance, go beyond tool capabilities. These requirements indicate that practitioners not only require a simple modeling tool that facilitates the application of a modeling language such as UML, but that practitioners need an RE approach that includes a tool. The study result confirmed the main challenges addressed by MIRA and was used as an input to develop the MIRA approach.

10.1.2.2 Development of the Model-based RE Approach

The thesis presented the MIRA approach that was developed based on the results from the first part of this thesis.

Contribution 3: An Artifact Reference Structure for Model-based Quality Assurance. This thesis contributes the MIRA artifact reference structure that integrates the RE concepts and QA quality factors resulting from the first study. Thereby, each of the model elements has a clear positive impact on the quality assurance of functional requirements. The artifact reference structure defines the concepts to be specified during RE. Attributes structure the information of each concept. MIRA distinguishes three RE concepts to describe functional requirements: Scenarios, interface requirements, and user functions. These RE concepts can be represented formally based on the FOCUS modeling theory. The MIRA artifact reference structure is embedded into the SPES modeling framework for system and software development.

Contribution 4: A Guideline for the Model-based Specification and Quality Assurance of Functional Requirements. A contribution of this thesis is the MIRA guideline that defines a coherent set of steps describing how to conduct the specification and quality assurance of functional requirements. The specification includes the textual specification and the formalization of functional requirements. Quality assurance activities comprise the analysis, validation and verification of functional requirements. The RE concepts, attributes and formal representations defined in the MIRA artifact reference structure facilitate syntactical analyses on the structure of the requirements specification. These checks can even detect some quality issues, when a requirement is documented as prose. A formal representation of a functional requirement facilitates formal analysis techniques such as checks on non-determinism. An executable representation enables a simulation of functional requirements.

10.1.2.3 Validation and Evaluation of the Model-based RE Approach

This thesis presented the technical validation of the MIRA approach in a feasibility analysis and its evaluation in case studies.

Contribution 5: Feasibility Analysis of the MIRA Approach. The thesis contributes a feasibility analysis of the MIRA approach through the implementation in a tool. The MIRA tool implements the MIRA artifact reference structure as a data model and offers operations on the data model to support the MIRA guideline. The MIRA tool facilitates data manipulation to create, delete, and change the requirements specification and parts thereof. The MIRA tool provides automated reviews on the structure of the requirements specification, formal techniques for the analysis and verification, and an animation-based simulation of functional requirements. Different views on the data, for example, a tree view on the documented trace links, visualize and filter the requirements specification based on the MIRA artifact reference structure. These views support manual reviews of the requirements specification.

Contribution 6: Case Study on the Effective Application of the MIRA Approach. The first case study applied the MIRA approach to develop a part of a train control system. The case study demonstrates that the MIRA approach is indeed applicable for an industrial specification. The case study applies MIRA to the door control of an automated train control system. Hence, the case study shows that the MIRA approach scales to the size of several user-visible system functions. The case study contains requirements for the train and for train controller subsystems. Therefore, the case study also demonstrates that MIRA is applicable to both system and subsystem level. The case study identifies findings in the requirements specification that substantiate some of the expected benefits of the MIRA artifact reference structure. Another result of this case study is an improvement of the MIRA approach, resulting in a refined guideline for the formalization of requirements and refinement links and in the introduction of interface requirements in the MIRA artifact reference structure.

The MIRA approach was applied in several other case studies. The second case study presented in this thesis demonstrated the extensibility of MIRA with a set of requirement types specific for a flight control system. The case study was based on an in-

dustrial specification from the domain of flight control systems. Further case studies were presented briefly; they indicate that the MIRA approach is also effectively applicable in different domains and by different user groups. Additionally, they show that the MIRA approach facilitates other researchers in this field to carry out research that builds on MIRA. For example, researchers investigated questions concerning seamless model-based development. Other researchers developed approaches for extra-functional requirements such as safety requirements.

10.1.3 Scientific Approach

In the following we discuss the scientific soundness of the MIRA approach.

Novelty. Cheng and Atlee stated in their paper on research directions in RE "Most research projects focus on a single RE problem, such as elicitation or traceability. As a result, the state of the art in RE research is a collection of technologies that have been researched and evaluated in isolation, with little knowledge of how to combine techniques effectively" [CA07]. To increase the "knowledge of how to combine techniques effectively" [CA07], MIRA uses well-known and proven RE concepts and model-based specification and quality assurance techniques. This thesis showed a systematic way to investigate combinations of these concepts and techniques with respect to a comprehensive and clear impact on quality assurance. Furthermore, this thesis showed how to integrate these techniques based on the study results. In addition, this thesis yielded new insights on the nature of RE quality factors (Chapter 4) and on the refinement of system to subsystem requirements (Chapter 9). With respect to the concrete approaches used in this thesis, MIRA closes a gap between these. Currently, the SPES modeling framework [PHAB12] focuses on supporting elicitation of requirements and communication between the stakeholders. MIRA defines the contents of the requirements viewpoint using the FOCUS system modeling theory [BS01] and provides method and tool support. The integration of SPES and FOCUS for the requirements viewpoint is new.

Effectiveness. MIRA is effective in the sense that each of the model elements in MIRA has a positive impact on the quality of requirements. This effectivity is addressed by construction through a detailed investigation and argumentation that each model element has a positive impact on the specification, analysis, validation or verification of functional requirements as investigated in the first study presented in this thesis. For example, the MIRA artifact reference structure contains a glossary entry, because defining glossary terms and their definition decreases term ambiguity. Ambiguous terms might influence the validation of requirements negatively, as different stakeholders could understand this term differently. Therefore, glossary terms have a positive impact on the validation of requirements. These impacts could be substantiated by literature to a varying degree of proof. The main positive impacts of MIRA were confirmed in a case study.

Well-grounded. Each of the model elements integrated in the MIRA approach is well-grounded, it references either to an established RE practice or is based on a log-

ical approach. The references are given in Chapter 6 that presents the MIRA artifact reference structure and in Chapter 7 that presents the guideline.

Relevance. The initial need for an approach like MIRA was stated in a survey conducted by Sikora et al. [STP12]. The authors pointed out that an adequate method and tool support for model-based RE that supports both the textual and formal representation of requirements is still missing. This need was confirmed and refined in a study into requirements for a model-based RE requirements that the author conducted in collaboration with Dongyue Mou and Maged Khalil. The results of this study are presented in Chapter 5.

Applicability and Scalability. This work showed that the MIRA approach can be applied to the specification and quality assurance of requirements. The MIRA guideline provides a theoretical validation of the applicability of the MIRA artifact reference structure. The MIRA tool provides mechanisms to apply the MIRA guideline and demonstrates the feasibility of the MIRA artifact reference structure. The case study on a train control system evaluates the applicability and scalability of the MIRA approach to functional requirements for a system function. Other case studies indicate that MIRA is also applicable for other domains. The case studies are presented in Chapter 9.

10.2 Outlook

This section presents the main open research questions that we identified during the development of the MIRA approach.

Extra-functional Requirements. The MIRA artifact reference structure has been extended with extra-functional requirements in case studies. Nevertheless, an activity-based investigation of extra-functional requirements with respect to quality assurance for the integration in the MIRA approach is still missing.

Systematic Handling of Modes. Modes are a means to describe dependencies between system functions. The theoretical foundation for modes in RE has been provided by Vogelsang [Vog15]. As the case study of a train control system showed, this theoretical foundation is compatible with the MIRA approach. Hence, a next logical step would be to extend the MIRA approach so that it incorporates the explicit handling of modes.

Further Automation. Automation could provide further potential savings. For example, the redundancy in the different representation forms could be handled by automated transformations from the formal representation to a textual representation of a requirement and vice versa. This automation promises savings, but at the

same time restricts the expressiveness of the less formal representation. Further automation could be the subject of detailed studies to establish its advantages and disadvantages.

Impacts of QA Quality Factors. This thesis investigated the positive impacts of QA quality factors on system quality factors through quality assurance. To strengthen the findings from the study results, further investigations on the impacts are necessary. Further research on the evidence for the positive impacts in different settings would improve the reliability of the study. Negative impacts should be investigated and balanced with the positive impacts. As the study was limited to a set of QA activities, the positive and negative impacts of the QA quality factors on other activities could be studied, for example regarding collaboration and communication.

Costs and Benefits. The benefit of the MIRA approach investigated in this thesis is the effective and efficient quality assurance of functional requirements. This thesis did not quantify this benefit, investigate further benefits, or relate the benefits of the approach to the costs of applying the MIRA approach. The author performed a first step towards an investigation of further benefits of the MIRA approach by investigating the impact of the granularity of an artifact reference structure on the change impact analysis [TH15].

Bibliography

[ABR05] Anne Angermann, Michael Beuschel, and Martin Rau. *Matlab - Simulink - Stateflow. Mit erweiterter CD-ROM. Grundlagen, Toolboxen, Beispiele*. Oldenbourg, 2005. (cited on p 50)

[AIP07] M. Autili, P. Inverardi, and P. Pelliccione. Graphical scenarios for specifying temporal properties: an automated approach. *Automated Software Engineering*, 2007. (cited on p 35)

[AVT+15] Vincent Aravantinos, Sebastian Voss, Sabine Teufl, Florian Hölzl, and Bernhard Schätz. Autofocus 3: Tooling concepts for seamless, model-based development of embedded systems. In *Proceedings of the 8th International Workshop on Model-based Architecting of Cyber-Physical and Embedded Systems (ACES-MB@MODELS)*, 2015. (cited on pp 10, 11, 29, 105, 153)

[BB06] Brian Berenbach and Gail Borotto. Metrics for model driven requirements development. In *Proceedings of the 28th International Conference on Software Engineering (ICSE)*, 2006. (cited on p 86)

[BBB+12] Jiri Barnat, Jan Beran, Lubos Brim, Tomas Kratochvíla, and Petr Ročkai. *Formal Methods for Industrial Critical Systems*, chapter Tool Chain to Support Automated Formal Verification of Avionics Simulink Designs, pages 78–92. Springer Berlin Heidelberg, 2012. (cited on p 46)

[BCG+10] Roderick Bloem, Alessandro Cimatti, Karin Greimel, Georg Hofferek, Robert Könighofer, Marco Roveri, Viktor Schuppan, and Richard Seeber. Ratsy - a new requirements analysis tool with synthesis. In *Computer Aided Verification*, 2010. (cited on p 44)

[BCMW15] John Backes, Darren Cofer, Steven Miller, and MichaelW. Whalen. Requirements analysis of a quad-redundant flight control system. In *NASA Formal Methods*. Springer International Publishing, 2015. (cited on pp 45, 46, 47, 48, 49)

[BCP+07] Roderick Bloem, Roberto Cavada, Ingo Pill, Marco Roveri, and Andrei Tchaltsev. Rat: A tool for the formal analysis of requirements. In *Computer Aided Verification*. Springer Berlin Heidelberg, 2007. (cited on p 44)

[BDH+12] Manfred Broy, Werner Damm, Stefan Henkler, Klaus Pohl, Andreas Vogelsang, and Thorsten Weyer. *Model-based Engineering of Embedded Systems: The SPES 2020 Methodology*, chapter Introduction to the SPES Modeling Framework, pages 31–49. Springer Berlin Heidelberg, 2012. (cited on pp 13, 22, 23, 24)

[Ber10] Brian Berenbach. Requirements engineering for industrial systems: No easy answers. In *Proceedings of the 18th IEEE International Requirements En-*

gineering Conference (RE), 2010. (cited on p 86)

[BFH+10] Manfred Broy, Martin Feilkas, Markus Herrmannsdoerfer, Stefano Merenda, and Daniel Ratiu. Seamless model-based development: From isolated tools to integrated model engineering environments. *Proceedings of the IEEE*, 2010. (cited on pp 3, 4, 22, 72, 154)

[BJV+14] Wolfgang Böhm, Maximilian Junker, Andreas Vogelsang, Sabine Teufl, Ralf Pinger, and Karsten Rahn. A formal systems engineering approach in practice: An experience report. In *Proceedings of the 1st International Workshop on Software Engineering Research and Industrial Practices (SER&IPs)*, 2014. (cited on pp 11, 131, 168, 176)

[BMR12] Jan Olaf Blech, Dongyue Mou, and Daniel Ratiu. Reusing test-cases on different levels of abstraction in a model based development tool. In *Proceedings of the Seventh Workshop on Model-Based Testing (MBT)*, 2012. (cited on pp 32, 35, 140, 164)

[BP88] Barry W. Boehm and Philip N. Papaccio. Understanding and controlling software costs. *IEEE Transactions on Software Engineering*, 1988. (cited on p 1)

[Bra08] Ian Brace. *Questionnaire Design: How to Plan, Structure and Write Survey Material for Effective Market Research (Market Research in Practice)*. Kogan Page, 2nd edition, September 2008. (cited on pp 85, 99)

[BRB+14] Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt. Challenges and practices in aligning requirements with verification and validation: A case study of six companies. *Empirical Software Engineering*, 2014. (cited on p 101)

[Bro06] Manfred Broy. Requirements engineering as a key to holistic software quality. In *Computer and Information Sciences – ISCIS*, 2006. (cited on p 100)

[Bro10a] Manfred Broy. A logical basis for component-oriented software and systems engineering. *The Computer Journal*, 2010. (cited on p 28)

[Bro10b] Manfred Broy. Multifunctional software systems: Structured modeling and specification of functional requirements. *Science of Computer Programming*, 2010. (cited on pp 9, 10, 22, 25, 115, 130)

[Bro13a] Manfred Broy. *Engineering Dependable Software Systems*, chapter A Logical Approach to Systems Engineering Artifacts and Traceability: From Requirements to Functional and Architectural Views, pages 1–48. IOS Press, 2013. (cited on pp 9, 22, 24, 25, 64, 76, 127)

[Bro13b] Manfred Broy. *Perspectives on the Future of Software Engineering: Essays in Honor of Dieter Rombach*, chapter Domain Modeling and Domain Engineering: Key Tasks in Requirements Engineering, pages 15–30. Springer Berlin Heidelberg, 2013. (cited on p 19)

[Bro15a] M. Broy. Rethinking nonfunctional software requirements. *Computer*, 2015. (cited on pp 14, 25)

[Bro15b] Manfred Broy. Logische und Methodische Grundlagen der Programm- und Systementwicklung. Vorlesungsskript, 2015. (cited on p 18)

[BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001. (cited on pp 5, 7, 9, 18, 21, 22, 25, 28, 29, 31, 46, 63, 73, 119, 197)

[BS10] Devesh Bhatt and Kirk Schloegel. Effective verification of flight critical software systems: Issues and approaches. *Presented at NSF/Microsoft Research Workshop on Usable Verification.*, 2010. (cited on pp 46, 50)

[BW81] Patrick Biernacki and Dan Waldorf. Snowball sampling: Problems and techniques of chain referral sampling. *Sociological methods & research*, 1981. (cited on p 42)

[CA07] B.H.C. Cheng and J.M. Atlee. Research directions in requirements engineering. In *Proceedings of the International Conference on the Future of Software Engineering (FOSE)*, 2007. (cited on pp 2, 15, 100, 197)

[CCGR00] Alessandro Cimatti, Edmund Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer (STTT)*, 2000. (cited on pp 35, 51)

[CDGNFA+12] Juan M. Carrillo De Gea, Joaquín Nicolás, José L. Fernández Alemán, Ambrosio Toval, Christof Ebert, and Aurora Vizcaíno. Requirements engineering tools: Capabilities, survey and assessment. *Inf. Softw. Technol.*, 2012. (cited on p 101)

[CGP99] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999. (cited on p 76)

[CHN11] A. Campetelli, F. Hölzl, and P. Neubeck. User-friendly model checking integration in model-based development. In *Proceedings of the 24th International Conference on Computer Applications in Industry and Engineering (CAINE)*, 2011. (cited on p 35)

[CJB+15] Alarico Campetelli, Maximilian Junker, Birthe Böhm, Maria Davidich, Vasileios Koutsoumpas, Xiuna Zhu, and Jan Christoph Wehrstedt. A model-based approach to formal verification in early development phases: A desalination plant case study. In *Gemeinsamer Tagungsband der Workshops der Tagung Software Engineering 2015, Fünfter Workshop zur Zukunft der Entwicklung softwareintensiver, eingebetteter Systeme (ENVISION 2020)*, 2015. (cited on p 190)

[Coc00] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., 2000. (cited on pp 9, 67, 109, 115, 116, 119, 120)

[Cona] Wikipedia Contributors. Ambiguity. Accessed April 30, 2015. (cited on p 63)

[Conb] Wikipedia Contributors. Kano model. Accessed June 1, 2015. (cited on p 108)

[CRST09] Alessandro Cimatti, Marco Roveri, Angelo Susi, and Stefano Tonetta. From informal requirements to property-driven formal validation. In *Formal Methods for Industrial Critical Systems*. Springer Berlin Heidelberg, 2009. (cited on pp 9, 36, 130, 150)

[CRST13] Alessandro Cimatti, Marco Roveri, Angelo Susi, and Stefano Tonetta. Validation of requirements for hybrid systems: A formal approach. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2013. (cited on pp 9, 36, 37, 45, 45, 46, 51, 130)

[DAC99] M.B. Dwyer, G.S. Avrunin, and J.C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st International Conference on Software Engineering (ICSE)*, 1999. (cited on p 31)

[DDR03] Emmanuelle Delor, Robert Darimont, and André Rifaut. Software quality starts with the modelling of goal-oriented requirements. In *Proceedings*

of the 16th International Conference Software & Systems Engineering and their Applications, 2003. (cited on pp 46, 49)

[DO112] DO-178C, software considerations in airborne systems and equipment certification, 2012. (cited on pp 101, 127)

[dSAVP10] Thiago C. de Sousa, Jorge R. Almeida, Jr., Sidney Viana, and Judith Pavón. Automatic analysis of requirements consistency with the b method. *SIGSOFT Software Engineering Notes*, 2010. (cited on pp 45, 46, 47, 65)

[DTW12] Marian Daun, Bastian Tenbergen, and Thorsten Weyer. *Model-based Engineering of Embedded Systems: The SPES 2020 Methodology*, chapter Requirements Viewpoint, pages 51–68. Springer Berlin Heidelberg, 2012. (cited on pp 23, 24, 59, 61, 66, 77, 78, 104, 105, 120, 124)

[EGHB07] Alexander Egyed, Paul Grünbacher, Matthias Heindl, and Stefan Biffl. Value-based requirements traceability: Lessons learned. In *Proceedings of the 21st IEEE International Requirements Engineering Conference (RE)*, 2007. (cited on p 85)

[Eis07] Cindy Eisner. Psl for runtime verification: Theory and practice. In *Proceedings of the 7th International Conference on Runtime Verification (RV)*, 2007. (cited on p 36)

[ESH14] Christian Ellen, Sven Sieverding, and Hardi Hungar. Detecting consistencies and inconsistencies of pattern-based functional requirements. In *Proceedings of the 19th International Conference on Formal Methods for Industrial Critical Systems (FMICS)*, 2014. (cited on pp 46, 50)

[EVMF16] Jonas Eckhardt, Andreas Vogelsang, and Daniel Méndez Fernández. Are non-functional requirements really non-functional? An investigation of non-functional requirements in practice. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, 2016. (cited on p 67)

[Eza15] Elijah Ezaga. Model-based requirements engineering: A systematic mapping study. Master thesis, Techische Universität München, Sep. 2015. Supervisor: Prof. Manfred Broy, Advisor: Sabine Teufl. (cited on pp 3, 39, 44)

[FCC13] Davide Falessi, Giovanni Cantone, and Gerardo Canfora. Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. *IEEE Transactions on Software Engineering*, 2013. (cited on p 61)

[FdS12] David de Almeida Ferreira and Alberto Rodrigues da Silva. Formally specifying requirements with rsl-il. In *Proceedings of the Eighth International Conference on the Quality of Information and Communications Technology (QUATIC)*, 2012. (cited on pp 45, 46)

[FFLS08] Fabrizio Fabbrini, Mario Fusani, Giuseppe Lami, and Edoardo Sivera. Software engineering in the european automotive industry: Achievements and challenges. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, 2008. (cited on pp 86, 100)

[FG12] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley Professional, 2012. (cited on pp 47, 73)

[FGZ15] Samuel A. Fricker, Rainer Grau, and Adrian Zwingli. *Requirements Engineering for Digital Health*, chapter Requirements Engineering: Best Practice, pages 25–46. Springer International Publishing, 2015. (cited on p 69)

[FHK+15] Stefan Feldmann, Sebastian J.I. Herzig, Konstantin Kernschmidt, Thomas Wolfenstetter, Daniel Kammerl, Ahsan Qamar, Udo Lindemann, Helmut Krcmar, Christiaan J.J. Paredis, and Birgit Vogel-Heuser. Towards effective management of inconsistencies in model-based engineering of automated production systems. *Proceedings of the15th IFAC Symposium on Information Control Problems in Manufacturing (INCOM)*, 2015. (cited on p 64)

[Fir05] Donald Firesmith. Are your requirements complete? *Journal of Object Technology*, 2005. (cited on p 65)

[FMF15] Henning Femmer, Jakob Mund, and Daniel Méndez Fernández. It's the activities, stupid! A new perspective on re quality. In *Proceedings of the IEEE/ACM 2nd International Workshop on Requirements Engineering and Testing (RET)*, 2015. (cited on pp 17, 57, 78)

[FMK+11] S. Farfeleder, T. Moser, A. Krall, T. Stålhane, H. Zojer, and C. Panis. DODT: Increasing requirements formalism using domain ontologies for improved embedded systems development. In *Proceedings of the IEEE 14th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS 2011)*, 2011. (cited on pp 45, 46, 47, 50)

[FP99] Iris Fahrmeir and Ludwig Pigeot. *Statistik. Der Weg zur Datenanalyse*. Springer, Heidelberg, 1999. (cited on p 86)

[FR07] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *Proceedings of the International Conference on the Future of Software Engineering (FOSE)*, 2007. (cited on pp 19, 100)

[FW13] Daniel Méndez Fernández and Stefan Wagner. Naming the pain in requirements engineering: Design of a global family of surveys and first results from germany. In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 2013. (cited on p 65)

[Gar12] Stefanie Gareis. Model-based development of a pacemaker, 2012. Bachelor's thesis. (cited on p 189)

[GBC+07] N. Gorse, P. Bélanger, A. Chureau, E.M. Aboulhamid, and Y. Savaria. A high-level requirements engineering methodology for electronic system-level design. *Computers & Electrical Engineering*, 2007. (cited on pp 45, 46)

[GBJ02] Martin Glinz, Stefan Berner, and Stefan Joos. Object-oriented modeling with ADORA. *Journal Information Systems*, 2002. (cited on p 44)

[GBR+00] Martin Glinz, Stefan Berner, Johannes Ryser, Stefan Joos, Nancy Schett, Reto Schmid, Yong Xia, and Robert Bosch Gmbh. The ADORA approach to object-oriented modeling of software. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE)*, 2000. (cited on pp 149, 150)

[GCHH+12a] Orlena Gotel, Jane Cleland-Huang, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, and Giuliano Antoniol. The quest for ubiquity: A roadmap for software and systems traceability research. In *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE)*, 2012. (cited on p 100)

[GCHH+12b] Orlena Gotel, Jane Cleland-Huang, JaneHuffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, Alex Dekhtyar, Giuliano Antoniol, Jonathan Maletic, and Patrick Mäder. Traceability fundamentals. In *Software and Systems Traceability*. Springer London, 2012. (cited on pp 9, 75, 118)

[GDTS07] Sébastien Gérard, Cédric Dumoulin, Patrick Tessier, and Bran Selic. Pa-

pyrus: A UML2 tool for domain-specific language modeling. In *Model-Based Engineering of Embedded Real-Time Systems - International Dagstuhl Workshop, Dagstuhl Castle. Revised Selected Papers*, 2007. (cited on p 51)

[GGJZ00] C.A. Gunter, E.L. Gunter, M. Jackson, and P. Zave. A reference model for requirements and specifications. *IEEE Software*, 2000. (cited on p 47)

[GGS06] Eva Geisberger, Johannes Grunbauer, and Bernhard Schatz. Interdisciplinary requirements analysis using the model-based RM tool AUTORAID. In *Proceedings of the International Automotive Requirements Engineering Workshop (AURE)*, 2006. (cited on pp 35, 46)

[GJ16] Fabien Gaucher and Bertrand Jeannet. Debugging real-time systems requirements with STIMULUS: A case-study from the automotive industry. White Paper, 2016. (cited on pp 44, 72)

[GKvdBV11] Arda Goknil, Ivan Kurtev, Klaas van den Berg, and Jan-Willem Veldhuis. Semantics of trace relations in requirements models for consistency checking and inferencing. *Software & Systems Modeling*, 2011. (cited on p 64)

[Gli07] M. Glinz. On non-functional requirements. In *Proceedings of the 15th IEEE International Requirements Engineering Conference (RE)*, 2007. (cited on p 67)

[Gro13] The Object Management Group. *Semantics of a Foundational Subset for Executable UML Models (FUML)*. Pearson Higher Education, 2013. (cited on p 44)

[HBY13] S. Hesari, R. Behjati, and T. Yue. Towards a systematic requirement-based test generation framework: Industrial challenges and needs. In *Proceedings of the 21st IEEE International Requirements Engineering Conference (RE)*, 2013. (cited on p 101)

[HD98] Patrick Heymans and Eric Dubois. Scenario-based techniques for supporting the elaboration and the validation of formal requirements. *Requirements Engineering*, 1998. (cited on pp 45, 46, 49)

[Hei07] Mats P. E. Heimdahl. Safety and software intensive systems: Challenges old and new. In *Proceedings of the International Conference on the Future of Software Engineering (FOSE)*, 2007. (cited on pp 45, 46, 48, 49, 76, 86, 100, 150)

[HF10] Florian Hölzl and Martin Feilkas. Autofocus 3: A scientific tool prototype for model-based development of component-based, reactive, distributed systems. In *Proceedings of the International Dagstuhl Conference on Model-based Engineering of Embedded Real-time Systems (MBEERTS)*, 2010. (cited on pp 10, 29)

[HHS07] James R. Hurford, Brendan Heasley, and Michael B. Smith. *Semantics: A Coursebook, 2nd ed.* Cambridge University Press, 2007. (cited on p 63)

[HJ07] Constance L. Heitmeyer and Ralph D. Jeffords. Applying a formal requirements method to three nasa systems: Lessons learned. In *In Proceedings of the IEEE Aerospace Conference*, 2007. (cited on p 44)

[HJL96] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 1996. (cited on pp 44, 65)

[HL96] M.P.E. Heimdahl and N.G. Leveson. Completeness and consistency in hierarchical state-based requirements. *IEEE Transactions on Software Engineering*, 1996. (cited on p 63)

[HPP+15] J. Holt, S. Perry, R. Payne, J. Bryans, S. Hallerstede, and F.O. Hansen. A model-based approach for requirements engineering for systems of systems. *IEEE Systems Journal*, 2015. (cited on p 44)

[IEE08] ISO/IEC/IEEE 12207-2008 Standard for Systems and Software Engineering – Software Life Cycle Processes, 2008. (cited on p 16)

[Ili07] D. Ilic. Deriving formal specifications from informal requirements. In *Proceedings of the 31nd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, 2007. (cited on pp 45, 46)

[IR12] Claire Ingram and Steve Riddle. Cost-benefits of traceability. In *Software and Systems Traceability*. Springer London, 2012. (cited on p 126)

[ISO09] ISO/IEC TR 24766:2009, Information technology – Systems and software engineering – Guide for requirements engineering tool capabilities, 2009. (cited on p 101)

[ISO10] ISO/IEC/IEEE 24765:2010, Systems and software engineering – Vocabulary, 2010. (cited on pp 2, 17)

[ISO11a] ISO 26262 Road vehicles – Functional safety, 2011. (cited on pp 85, 101)

[ISO11b] ISO/IEC/IEEE 29148:2011(E) Systems and software engineering – Life cycle processes –Requirements engineering, 2011. (cited on pp 16, 62, 75, 78, 104)

[ITU11a] Recommendation Z.120 (02/11): Message Sequence Chart (MSC) - Language Requirements and Framework, February 2011. approved February 2011. Editor: D. Amyot. (cited on pp 22, 31)

[ITU11b] Recommendation Z.150 (02/11): User Requirements Notation (URN) - Language Requirements and Framework, February 2011. approved February 2011. Editor: D. Amyot. (cited on p 101)

[Jac92] Ivar Jacobson. *Object Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Professional, 1992. (cited on p 44)

[Jac95] Michael Jackson. The world and the machine. In *Proceedings of the 17th International Conference on Software Engineering (ICSE)*, 1995. (cited on pp 13, 14)

[JDF+10] Elmar Juergens, Florian Deissenboeck, Martin Feilkas, Benjamin Hummel, Bernhard Schaetz, Stefan Wagner, Christoph Domann, and Jonathan Streit. Can clone detection support quality assessments of requirements specifications? In *Proceedings of the 32nd International Conference on Software Engineering (ICSE)*, 2010. (cited on p 68)

[JFH91] W. Lewis Johnson, Martin S. Feather, and David R. Harris. Integrating domain knowledge, requirements, and specifications. *Journal of Systems Integration*, 1991. (cited on pp 46, 48, 49)

[JHLR10] Michael Jastram, Stefan Hallerstede, Michael Leuschel, and Aryldo G. Russo, Jr. An approach of requirements tracing in formal refinement. In *Proceedings of the Third International Conference on Verified Software: Theories, Tools, Experiments (VSTTE)*, 2010. (cited on pp 45, 46, 47)

[JLHM91] M.S. Jaffe, N.G. Leveson, M.P.E. Heimdahl, and B.E. Melhart. Software requirements analysis for real-time process-control systems. *IEEE Transactions on Software Engineering*, 1991. (cited on p 65)

[JMM99] Natalia Juristo, José L Morant, and Ana M. Moreno. A formal approach for generating oo specifications from natural language. *Journal of Systems and Software*, 1999. (cited on pp 45, 46)

[JN12] Maximilian Junker and Philipp Neubeck. A rigorous approach to availability modeling. In *Proceedings of the 4th International Workshop on Modeling in Software Engineering (MiSE)*, 2012. (cited on p 15)

[Jus13] Benjamin Justice. Natural language specifications for safety-critical systems. Master's thesis, Carl von Ossietzky Universität Oldenburg, 2013. (cited on pp 46, 48, 50)

[KC05a] S. Konrad and B.H.C. Cheng. Facilitating the construction of specification pattern-based properties. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE)*, 2005. (cited on pp 49, 150)

[KC05b] Sascha Konrad and Betty H. C. Cheng. Real-time specification patterns. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, 2005. (cited on pp 45, 46, 47)

[KC07] Barbara Kitchenham and Stuart Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007. (cited on p 83)

[KJL09] P. Kroha, R. Janetzko, and J.E. Labra. Ontologies in checking for inconsistency of requirements specification. In *Proceedings of the Third International Conference on Advances in Semantic Processing (SEMAPRO)*, 2009. (cited on pp 45, 45, 46, 47)

[Kof10] Leonid Kof. From requirements documents to system models: A tool for interactive semi-automatic translation. In *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE)*, 2010. (cited on p 46)

[KP11] Leonid Kof and Birgit Penzenstadler. From requirements to models: Feedback generation as a result of formalization. In *Advanced Information Systems Engineering*. Springer Berlin Heidelberg, 2011. (cited on pp 45, 46, 47, 49)

[KRSV13] Antoaneta Kondeva, Daniel Ratiu, Bernhard Schätz, and Sebastian Voss. Seamless model-based development of embedded systems with AF3 phoenix. In *Proceedings of the 20th IEEE International Conference and Workshops on the Engineering of Computer Based Systems (ECBS)*, 2013. (cited on p 29)

[KS98] Gerald Kotonya and Ian Sommerville. *Requirements Engineering - Processes and Techniques*. John Wiley & Sons, 1998. (cited on pp 14, 15, 16, 78, 107, 108)

[KS04] Hye Yeon Kim and Frederick T. Sheldon. Testing software requirements with z and statecharts applied to an embedded control systemt0t1. *Software Quality Journal*, 2004. (cited on pp 46, 47)

[LCK98] Woo Jin Lee, Sung Deok Cha, and Yong Rae Kwon. Integration and analysis of use cases using modular petri nets in requirements engineering. *IEEE Transactions on Software Engineering*, 1998. (cited on pp 45, 46)

[LSS94] O. I. Lindland, G. Sindre, and A. Solvberg. Understanding quality in conceptual modeling. *IEEE Software*, 1994. (cited on p 78)

[LvL02] Emmanuel Letier and Axel van Lamsweerde. Deriving operational software specifications from system goals. *SIGSOFT Software Engineering Notes*, 2002. (cited on pp 46, 48, 49, 49)

[MF11] Daniel Méndez Fernández. *Requirements Engineering: Artefact-Based Customisation*. PhD thesis, Institut für Informatik, Technische Universität München, 2011. (cited on pp 5, 20, 78, 104)

[MFBT] D. Méndez Fernández, M. Broy, and S. et al. Teufl. What is an artefact? Unpublished Draft, 27th of February 2015. (cited on p 20)

[MFP14] Daniel Méndez Fernández and Birgit Penzenstadler. Artefact-based requirements engineering: the AMDiRE approach. *Requirements Engineering*, 2014. (cited on pp 19, 20, 104, 104)

[MFPKB10] Daniel Méndez Fernández, Birgit Penzenstadler, Marco Kuhrmann, and Manfred Broy. A meta model for artefact-orientation: Fundamentals and lessons learned in requirements engineering. In *Model Driven Engineering Languages and Systems*. Springer Berlin / Heidelberg, 2010. (cited on p 19)

[MMFFE15] J. Mund, D. Mendez Fernandez, H. Femmer, and J. Eckhardt. Does quality of requirements specifications matter? combined results of two empirical studies. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2015. (cited on pp 7, 62)

[MMGD10] I. Menzel, M. Mueller, A. Gross, and J. Doerr. An experimental comparison regarding the completeness of functional requirements specifications. In *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE)*, 2010. (cited on p 76)

[MR12] Dongyue Mou and Daniel Ratiu. Binding requirements and component architecture by using model-based test-driven development. In *Proceedings of the IEEE First International Workshop on theTwin Peaks of Requirements and Architecture (Twin Peaks)*, 2012. (cited on pp 32, 35, 49, 164)

[MS03] Raimundas Matulevičius and Darijus Strašunskas. Evaluation framework of requirements engineering tools for verification and validation. In *Advanced Conceptual Modeling Techniques*. Springer Berlin / Heidelberg, 2003. (cited on pp 85, 101)

[MTWH06] Steven P. Miller, Alan C. Tribble, Michael W. Whalen, and Mats P. E. Heimdahl. Proving the shalls: Early validation of requirements through formal methods. *International Journal on Software Tools for Technology Transfer (STTT)*, 2006. (cited on pp 45, 46, 48, 49, 50, 73, 150)

[MW10] A. Mavin and P. Wilkinson. Big ears (the return of "easy approach to requirements engineering"). In *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE)*, 2010. (cited on pp 21, 76)

[MWHN09] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak. Easy approach to requirements syntax (ears). In *Proceedings of the 17th IEEE International Requirements Engineering Conference (RE)*, 2009. (cited on p 113)

[NE00] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the International Conference on the Future of Software Engineering (FOSE)*, 2000. (cited on p 100)

[NEA12] P.B.F. Njonko and W. El Abed. From natural language business requirements to executable models via sbvr. In *Proceedings of the International Conference on Systems and Informatics (ICSAI)*, 2012. (cited on pp 45, 46, 47)

[NER00] Bashar Nuseibeh, Steve Easterbrook, and Alessandra Russo. Leveraging inconsistency in software development. *Computer*, 2000. (cited on p 65)

[Neu12] Philipp Neubeck. *A Probabilitistic Theory of Interactive Systems*. Dissertation, Technische Universität München, München, 2012. (cited on p 15)

[NFTJ06] C. Nebut, F. Fleurey, Y. Le Traon, and J. M. Jezequel. Automatic test generation: a use case driven approach. *IEEE Transactions on Software Engineering*, 2006. (cited on p 77)

[OMG06] OMG Object Constraint Language (OMG OCL): OMG available specification Version 2.0, 2006. (cited on p 72)

[OMG11] OMG Unified Modeling Language (OMG UML), Superstructure v2.4.1, OMG document number formal/2011-08-06, 2011. (cited on pp 44, 104)

[OMG12] OMG Systems Modeling Language (OMG SysML) v1.3. OMG document number formal/2012-06-01, 2012. (cited on p 44)

[OMG13] OMG Requirements Interchange Format (ReqIF), v1.1, October 2013. (cited on p 165)

[PBKS07] A. Pretschner, M. Broy, I.H. Kruger, and T. Stauner. Software engineering for automotive systems: A roadmap. In *Proceedings of the International Conference on the Future of Software Engineering (FOSE)*, 2007. (cited on pp 86, 100)

[PDC+11] Paulo F. Pires, Flávia C. Delicato, Raphael Cóbe, Thais Batista, Joseph G. Davis, and Joo Hee Song. Integrating ontologies, model driven, and cnl in a multi-viewed approach for requirements engineering. *Requirements Engineering*, 2011. (cited on pp 45, 46, 47)

[PE12] B. Penzenstadler and J. Eckhardt. A requirements engineering content model for cyber-physical systems. In *Proceedings of the IEEE Second Workshop on Requirements Engineering for Systems, Services and Systems-of-Systems (RES4)*, 2012. (cited on pp 61, 104, 105)

[PFMM08] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 2008. (cited on p 39)

[PHAB12] Klaus Pohl, Harald Hönninger, Reinhold Achatz, and Manfred Broy, editors. *Model-based Engineering of Embedded Systems: The SPES 2020 Methodology*. Springer, 2012. (cited on pp 9, 23, 109, 110, 197)

[PMHP12] Amalinda Post, Igor Menzel, Jochen Hoenicke, and Andreas Podelski. Automotive behavioral requirements expressed in a specification pattern system: a case study at BOSCH. *Requirements Engineering*, 2012. (cited on pp 46, 47)

[Poh10] Klaus Pohl. *Requirements Engineering - Fundamentals, Principles, and Techniques*. Springer, 2010. (cited on pp 9, 68, 106)

[RG11] S. Ramesh and A. Gadkari. Rigorous model-based design & verification flow for in-vehicle software. In *Proceedings of the 48th Design Automation Conference (DAC)*, 2011. (cited on pp 46, 48)

[RH09] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 2009. (cited on p 167)

[RHR+06] James L. Rash, Michael G. Hinchey, Christopher A. Rouff, Denis Gracanin, and John Erickson. A requirements-based programming approach to developing a nasa autonomous ground control system. *Artificial Intelligence Review*, 2006. (cited on pp 46, 49)

[RR06] Suzanne Robertson and James Robertson. *Mastering the Requirements Process (2nd Edition)*. Addison-Wesley Professional, 2006. (cited on pp 14, 68, 74, 104, 106, 107, 108, 109)

[RRH93] A.P. Ravn, H. Rischel, and K.M. Hansen. Specifying and verifying require-

ments of real-time systems. *IEEE Transactions on Software Engineering*, 19(1), 1993. (cited on p 151)

[RSF14] Alessandro Gerlinger Romero, Klaus Schneider, and Mauricio Goncalves Vieira Ferreira. Using the base semantics given by fUML for verification. In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2014. (cited on p 44)

[RTVH15] Susanne Rösch, Sabine Teufl, and Birgit Vogel-Heuser. Model-based quality assurance in machine and plant automation using sequence diagrams - a comparison of two research approaches. In *Proceedings of the 13th International Conference on Industrial Informatics (INDIN)*, 2015. (cited on pp 11, 131)

[SB13] Florian Schneider and Brian Berenbach. A literature survey on international standards for systems requirements engineering. *Procedia Computer Science*, 2013. (cited on p 62)

[SBC10] A. Sanyal, S.S. Basu, and S. Choudhury. A requirement framework for enablement of automatic generation of domain model. In *Proceedings of the International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, 2010. (cited on pp 45, 46, 47)

[SBHW03] B. Schätz, P. Braun, F. Huber, and A. Wisspeintner. Consistency in model-based development. In *Proceedings of the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, 2003. (cited on p 65)

[Sch09] Bernhard Schätz. *Model-Based Development of Software Systems: From Models to Tools.* Habilitation thesis, Technische Universität München, 2009. (cited on pp 9, 19, 29, 35, 36, 46, 59, 72, 76, 78, 104, 130, 143)

[Sch10] Rolf Schwitter. Controlled natural languages for knowledge representation. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING): Posters*, 2010. (cited on pp 21, 76)

[SD15] Souvik Sengupta and Ranjan Dasgupta. Use of semi-formal and formal methods in requirement engineering of ILMS. *SIGSOFT Software Engineering Notes*, 2015. (cited on pp 46, 47)

[SFGP05] Bernhard Schätz, Andreas Fleischmann, Eva Geisberger, and Markus Pister. Model-based requirements engineering with AUTORAID. In *Proceedings of INFORMATIK*, 2005. (cited on pp 9, 35, 46, 49)

[SJV12] Valdivino Alexandre de Santiago Júnior and Nandamudi Lankalapalli Vijaykumar. Generating model-based test cases from natural language requirements for space application software. *Software Quality Journal*, 2012. (cited on pp 45, 46, 47, 49)

[Sof11] Rational Software. Rational unified process best practices for software development teams. Technical report, Rational Software White Paper, TP026B, Rev 11/01, 2011. (cited on p 44)

[Som11] Ian Sommerville. *Software Engineering*. Pearson Education, 9th edition, 2011. (cited on p 14)

[SRB+00] Reto Schmid, Johannes Ryser, Stefan Berner, Martin Glinz, Ralf Reutemann, and Erwin Fahr. A survey of simulation tools for requirements engineering. Technical report, 2000. (cited on pp 86, 101)

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, Wien, 1973.

(cited on p 18)

[STP11] Ernst Sikora, Bastian Tenbergen, and Klaus Pohl. Requirements engineering for embedded systems: An investigation of industry needs. In *Proceedings of the 17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, 2011. (cited on pp 85, 101, 102)

[STP12] Ernst Sikora, Bastian Tenbergen, and Klaus Pohl. Industry needs and research directions in requirements engineering for embedded systems. *Requirements Engineering*, 2012. (cited on pp 3, 4, 81, 101, 198)

[SW14] T. Stalhane and T. Wien. The DODT tool applied to sub-sea software. In *Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE)*, 2014. (cited on pp 46, 50)

[SWN+08] D. Streitferdt, G. Wendt, P. Nenninger, A. Nyssen, and H. Lichter. Model driven development challenges in the automation domain. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, 2008. (cited on pp 86, 100)

[TBB+13] Bastian Tenbergen, Peter Battram, Thomas Buell, Martin Hiller, Mark Rzepka, Thorsten Weyer, and Philipp Bohn. Description of the requirements quality assessment framework. Technical Report Version 1.1, 2013. (cited on p 78)

[TBP14] S. Teufl, W. Böhm, and R. Pinger. Understanding and closing the gap between requirements on system and subsystem level. In *Proceedings of the 4th IEEE International Model-Driven Requirements Engineering Workshop (MoDRE)*, pages 77–86, Aug 2014. (cited on pp 11, 131, 151, 168, 180)

[TG14] Saurabh Tiwari and Atul Gupta. Does increasing formalism in the use case template help? In *Proceedings of the 7th India Software Engineering Conference (ISEC)*, 2014. (cited on p 75)

[TH15] Sabine Teufl and Georg Hackenberg. Efficient impact analysis of changes in the requirements of manufacturing automation systems. In *Proceedings of the15th IFAC Symposium on Information Control Problems in Manufacturing (INCOM)*, 2015. (cited on pp 11, 105, 126, 199)

[TKM13] Sabine Teufl, Maged Khalil, and Dongyue Mou. Requirements for a model-based requirements engineering tool for embedded systems: Systematic literature review and survey. White paper, 2013. (cited on pp 11, 85)

[TMR13] Sabine Teufl, Dongyue Mou, and Daniel Ratiu. Mira: A tooling-framework to experiment with model-based requirements engineering. In *RE*, 2013. (cited on pp 11, 153)

[VEFR12] Andreas Vogelsang, Sebastian Eder, Martin Feilkas, and Daniel Ratiu. *Model-based Engineering of Embedded Systems: The SPES 2020 Methodology*, chapter Functional Viewpoint, pages 69–83. Springer Berlin Heidelberg, 2012. (cited on p 114)

[VEH+14] Andreas Vogelsang, Sebastian Eder, Georg Hackenberg, Maximilian Junker, and Sabine Teufl. Supporting concurrent development of requirements and architecture: A model-based approach. In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2014. (cited on pp 11, 144, 190)

[vL01] A. van Lamsweerde. Goal-oriented requirements engineering: a guided tour. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, 2001. (cited on pp 19, 45, 46, 48, 49, 55, 66, 67, 110)

[vL03] A. van Lamsweerde. From system goals to software architecture. *Formal Methods for Software Architectures*, 2003. (cited on pp 46, 48, 49, 49)

[vL09] Axel van Lamsweerde. *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2009. (cited on pp 4, 9, 13, 14, 15, 15, 16, 58, 61, 62, 63, 64, 65, 69, 72, 80, 132, 164)

[Vog15] Andreas Vogelsang. *Model-based Requirements Engineering for Multifunctional Systems*. PhD thesis, Institut für Informatik, Technische Universität München, 2015. (cited on pp 168, 182, 198)

[VSKC13] Sebastian Voss, Bernhard Schätz, Maged Khalil, and Carmen Carlan. Towards modular certification using integrated model-based safety cases. In *VeriSure: Verification and Assurance Workshop*, 2013. (cited on p 189)

[Wag07] Stefan Wagner. *Cost-Optimisation of Analytical Software Quality Assurance*. PhD thesis, Institut für Informatik, Technische Universität München, 2007. (cited on p 57)

[WHR14] J. Whittle, J. Hutchinson, and M. Rouncefield. The state of practice in model-driven engineering. *IEEE Software*, 2014. (cited on p 49)

[WMMR05] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Requirements Engineering*, 2005. (cited on pp 42, 52)

[WPS02] M. Woodside, D. Petriu, and K. Siddiqui. Performance-related completions for software specifications. In *Proceedings of the 24rd International Conference on Software Engineering (ICSE)*, 2002. (cited on p 64)

[ZG02] Didar Zowghi and Vincenzo Gervasi. The three Cs of requirements: Consistency, completeness, and correctness. In *Proceedings of the 8th International Workshop on Requirements Engineering: Foundation for Software Quality, (REFSQ)*, 2002. (cited on p 65)

[ZG03] Didar Zowghi and Vincenzo Gervasi. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software Technology*, 2003. (cited on pp 62, 63)

[ZJ97] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 1997. (cited on pp 9, 111, 126, 136, 144)