Building Models We Can Rely On: Requirements Traceability for Model-Based Verification Techniques

Marco Filax(✉), Tim Gonschorek, and Frank Ortmeier

Chair of Software Engineering, Otto-von-Guericke University of Magdeburg, Magdeburg, Germany
{marco.filax,tim.gonschorek,frank.ortmeier}@ovgu.de

Abstract. Proving the safety of a critical system is a complex and complicated task. Model-based formal verification techniques can help to verify a System Requirement Specification (SRS) with respect to normative and safety requirements. Due to an early application of these methods, it is possible to reduce the risk of high costs caused by unexpected, late system adjustments. Nevertheless, they are still rarely used. One reason among others is the lack of an applicable integration method in an existing development process.

In this paper, we propose a process to integrate formal model-based verification techniques into the development life-cycle of a safety critical system. The core idea is to systematically refine informal specifications by (1) categorization, (2) structural refinement, (3) expected behavioral refinement, and finally, (4) operational semantics. To support modeling, traceability is upheld through all refinement steps and a number of consistency checks are introduced.

The proposed process has been jointly developed with the German Railroad Authority (EBA) and an accredited safety assessor. We implemented an Eclipse-based IDE with connections to requirement and systems engineering tools as well as various verification engines. The applicability of our approach is demonstrated via an industrial-sized case study in the context of the European Train Control System with ETCS Level 1 Full Supervision.

Keywords: Traceability · Verification · Practical experiences

1 Introduction

Developing safety critical systems is a complex and complicated task because malfunction imposes high costs or even endangers human lives. Therefore, safety critical systems are specified with an increasing amount of functional and non-functional requirements, to reduce the risk of malfunction and hazardous behavior.

Further, it has to be ensured that the system adheres to specific safety norms and standards, e.g., EN 50128 for railway, DO-178C for avionics or ISO 26262 for automotive applications. Before the developed safety critical system can be put into operation, it must be certified by a governmental authority according to the relevant safety norm. It must be shown that all normative and safety requirements have been met for the system.

The fulfillment of safety requirements covers the complete safety lifecycle of the system. It is desirable to detect possible malfunction and hazardous behavior as early as possible because unexpected, late system adjustments would impose even higher costs. To detect hazardous behavior a complete, consistent and correct System Requirement Specification (SRS) that adheres to all normative and safety requirements, is required. However, proving the fulfillment of all these requirements is a challenging task.

Formal verification techniques can help proving normative and safety requirements. Thus, almost all norms recommend the use of formal verification techniques to prove functional safety properties [9,15,16] depending on the required safety integrity level (SIL). The application of formal methods during the requirement specification phase is recommended for systems with SIL 2 and higher and highly recommended for systems with SIL 4 [15].

However, IEC 61508 acknowledges[1] that using formal methods may be challenging [15]. In our point of view, one reason is that none of these norms suggest an integration into the lifecycle. Additionally, they do not emphasize requirements traceability during the implementation of the formal model. As traceability is an issue for traditional methods, it is an enormous problem for formal verification techniques: formal models, which describe the complete behavior of a safety critical system mathematically, might be challenging to understand by others without a strong formal background. In our point of view, this increases the hurdle for applying formal verification techniques during the requirement specification phase. However, if every element of the formal model would be linked to the requirements, this hurdle is negotiated.

We collaborated with the German Federal Railroad Authority (EBA) and an accredited assessor to develop a structured workflow supporting safety engineers in integrating formal model-based verification techniques into the safety lifecycle of critical systems. The core idea is to systematically refine informal specifications by (1) categorization, (2) structural refinement, (3) exemplary behavioral refinement, and finally, (4) operational semantics. We rely on our previous work [11] on transforming informal natural language requirement via semi-formal UML models into a formal interpretation. Traceability is upheld through all refinement phases with the explicit need for linking requirements (e.g. the resulting formal model). We introduce some (semi-)automatic consistency checks and provide adequate tool support for the proposed process. Developing the formal model with the proposed process, which supports the usage of formal methods in a traceable, well-defined, reliable and understandable manner, increases the acceptance of the formal verification results.

In the following section, we present an overview of the proposed process and describe how to integrate it into a typical development process. We rely on

[1] Acknowledged in IEC 61508-7 section B2.2.

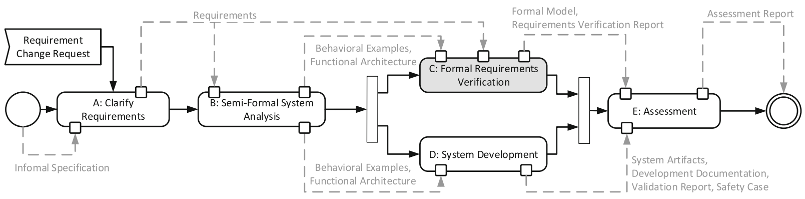

Requirement Change Request
Requirements
Formal Model, Requirements Verification Report
Assessment Report
Behavioral Examples, Functional Architecture
A: Clarify Requirements
B: Semi-Formal System Analysis
C: Formal Requirements Verification
D: System Development
E: Assessment
Infomal Specification
Behavioral Examples, Functional Architecture
System Artifacts, Development Documentation, Validation Report, Safety Case

Fig. 1. Five phases of the safety lifecycle: the phases *A*, *B*, *D*, *E* represent state-of-the-art actives. In this paper, we integrate phase *C: Formal Requirements Verification* into the development process. Solid lines represent control flow whereas dashed lines represent object flow.

state-of-the-art methods to determine a semi-formal UML model (cf. Sect. 3) based on the SRS. We then summarize the different steps to derive the formal model from the UML and the requirements in Sect. 4. Throughout the whole paper, we demonstrate the feasibility of the approach on a case study. At the end of this paper, we point out future directions.

2 Building Reliable Formal Models

The proposed approach to integrating formal verification techniques into the safety lifecycle is summarized in Fig. 1. Given a SRS we distinguish five phases: *A: Clarify Requirements*, *B: Semi-Formal System*, *C: Formal Requirements Verification*, *D: System Development*, and *E: Assessment*. The phases A, B, D, and E summarize state-of-the-art activities to develop a safety critical system. In the following we describe how to integrate phase *C: Formal Requirement Verification* in this typical development process. We also summarize activities of the other phases and specify extensions if necessary.

The different phases of the proposed process are implemented in the Verification Environment for Critical Systems (VECS). VECS supports the efficient development of formal models and gives the possibility to check the formal models in different model checking tools, without switching from one to another. The results can be analyzed directly in VECS. Further, we developed a plugin for a state-of-the-art UML modeling tool to enhance the integration in our formal verification tool. VECS enables the modeler to trace generated elements directly to the UML model what helps to validate that the generated formal elements are corresponding to the UML model. Additionally, we implemented traceability reports to analyze and document the coverage and linkage of requirements in UML.

We evaluate the proposed approach by formalizing the SRS of the European Train Control System (ETCS) Level 1 Full Supervision. For this, we modeled all requirements necessary for the verification corresponding to a reference track defined by the EBA[2]. This contains the communication via Eurobalises, locking

[2] Available online: https://cse.cs.ovgu.de/vecs/index.php/product/achievements/casestudies/etcs.

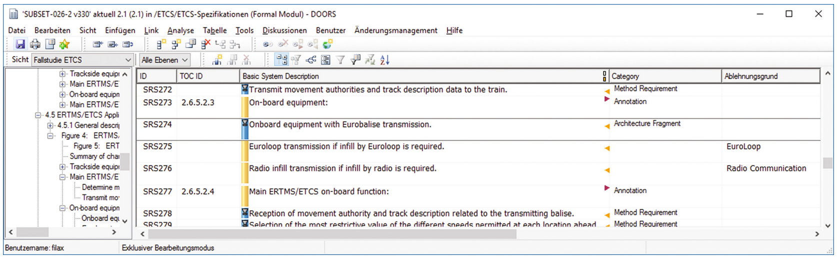
and releasing of track sections via Movement Authorities, the observance of partially overlaid speed restrictions as well as mode changes of the onboard unit. We will use the case study as a running example for the proposed approach.

2.1 A: Clarify Requirements

Informal requirements often are neither complete nor organized in a way that they can be processed in a structured manner. Hence, a system analyst must clarify the requirements and therefore split them into atomic statements as illustrated in Fig. 1. These atomic statements can be sorted into different categories, defined by a requirement pattern, to support following phases. There already exists a variety of successfully applied requirement patterns [20]. However, the final domain-specific needs of the system typically require an adoption of the used pattern.

A requirement pattern decreases the complexity of following phases for two reasons: On the one hand, the set of requirements becomes structured since for certain phases only specific categories are required (e.g. for the functional architecture we only need architectural requirements). On the other hand, the set of requirements becomes reduced since particular requirements could occur in several statements but can be combined into a single atomic statement.

As the system analyst splits requirements into atomic statements and categorizes them afterward, changes need to be documented. Changes in the requirements might occur in later phases of the process, e.g., when an error in the specification is detected and needs to be corrected. Such change requests may occur at any process phase (cf. *Requirement Change Request* in Fig. 1).



ID	TOC ID	Basic System Description	Category	Ablehnungsgrund
SRS272		Transmit movement authorities and track description data to the train.	Method Requirement	
SRS273	2.6.5.2.3	On-board equipment:	Annotation	
SRS274		Onboard equipment with Eurobalise transmission.	Architecture Fragment	
SRS275		Euroloop transmission if infill by Euroloop is required.		EuroLoop
SRS276		Radio infill transmission if infill by radio is required.		Radio Communication
SRS277	2.6.5.2.4	Main ERTMS/ETCS on-board function:	Annotation	
SRS278		Reception of movement authority and track description related to the transmitting balise.	Method Requirement	
SRS279		Selection of the most restrictive value of the different speeds permitted at each location ahead	Method Requirement	

Fig. 2. An excerpt of requirements from the functional specification of the ETCS. Note that these requirements have been clarified: Requirement 2.6.5.2.3 was split into four atomic statements. The requirements have been categorized and some are rejected as they are not in the scope of the case study.

We applied the requirement pattern presented in [8], adopted in our previous work [10], because it had been applied to this domain successfully. Our pattern contains eight categories: *Method*, *Sequence*, *User*, *State* and *Safety requirements*, *Architecture* and *Glossary fragments* and *Annotations*. Figure 2 depicts an exemplary application of the requirement pattern. The requirements are already clarified and split into atomic statements. We examined the functional description of the ETCS (SUBSET-26) with more than 33.000 requirements. We had to reduce the number of requirements which had to be formalized to handle the case study with the available manpower. We restricted us to ETCS Level 1, shown in Fig. 2 as the rejection of Euroloop and Radio Communication. From this, over 4.200 requirements had been identified as relevant for the case study.

2.2 B: Semi-Formal System Analysis

The *Semi-Formal System Analysis* represents typical system modeling tasks. It aims at modeling architectural and behavioral requirements in UML [17]. These modeling tasks are described in more detail in Sect. 3.

The overall idea is to determine a static functional architecture from the SRS. This is based on the observation that the SRS typically covers a basic structural definition of the system. Additionally, the SRS also typically covers a description of the intended behavior. Sequence diagrams are extracted from the behavioral requirements with the help of the functional architecture to represent the intended behavior. We focus on sequence diagrams because the SRS does not cover a complete behavioral definition. This is justified by the fact that a SRS is written in natural language.

We apply state-of-the-art requirements traceability: we use requirement diagrams to clarify the lifeline of requirements. Every element, derived from a requirement, shall be linked it. We use trace relations to link semi-formal elements and requirements. An example is depicted in Fig. 4b. Tool support is available in the form of Sparx Enterprise Architect with an IBM Rational DOORS connector.

2.3 C: Formal Verification

The goal of this phase is to verify safety claims. These are typically part of the SRS: we call them safety requirements. In order the verify these safety requirements with the help of model checking, we firstly need to build a formal model. We propose to develop the formal model using the semi-formal model. To do so, we reuse the semi-formal architecture: we automatically formalize the semi-formal architecture, by translating architectural elements to the specific formal language. The formal behavior has to be implemented manually. Further, we propose to use semi-formal sequences as an acceptance test for the formal model: we use them to generate acceptor automata in order to demonstrate that the behavior is implemented correctly in the formal model.

The formal model's architecture is automatically generated from the previous defined semi-formal architecture using the methods described in [11]. The basic idea is to translate every semi-formal architectural element into its formal equivalent, such that the transformation is bijective. We automatically derive requirement links to formal elements in order to provide traceability. The automatic transformation ensures the correctness of the mapping from the static architecture to the formal model.

When the architecture of the formal model is generated, the formal behavior has to be implemented as different automata. The modeler implements every automaton of the formal model with the help of a subset of requirements. The modeler has to preserve traceability manually by linking the requirements to the automata elements.

Since the automata are modeled by hand, it is important to demonstrate that they describe the intended system behavior. We use the previously defined sequences to automatically generate acceptance tests in the form of observer automata. By checking if the formal model fulfills the specified sequence we demonstrate the correct implementation of the expected behavior. Traceability is preserved by automatically linking the requirements linked to a sequence and a sequence ID to the generated observer.

Finally, given safety requirements are manually translated into temporal logic statements. We use these statements and formal model to verify the safety properties with the help of model checking [4]. The results are composed to a requirement verification report. We explain the complete phase in Sect. 4 in more detail.

2.4 D: System Development

Parallel to the phase C, “normal” system development activities are performed. We summarize these activities as *System Development* (cf. Fig. 1). These activities contain, e.g., the refinement of the functional architecture, the definition of the system design, the implementation of the system, and the verification and validation of the resulting implementation.

Phase D is not in the scope of this paper, as the process is meant to be an extension, at most to the specification phase of the safety lifecycle. However, we see the formal model as a possibility to aid in the verification and validation of the implementation. For example, different authors proposed approaches to utilize a model checker as a test oracle for the resulting implementation [3].

2.5 E: Assessment

Finally, the *Assessment* is performed - the overall functional safety is evaluated according to the relevant safety norm. Typical assessment documents like the validation report or safety cases are issued. Additionally, we issued a requirement verification report in phase C (cf. Sect. 2.3). It contains results, calculated by a model checker, which prove the safety requirements with mathematical rigor. The proves are used by an assessor to evaluate the functional safety of the system. However, the results generated from the model checker rely on the correct implementation of the formal model. If the formal model is faulty one has to question the result’s value for the assessment. This, in our opinion, demonstrates the importance of the proposed process: we have to ensure the correct implementation of the formal model. We introduced a set of semi-automatic checks to verify that the formal model implements the expected behavior (cf. Sect. 2.2). We also introduced a set of manually traceability reports, to validate that the automata rely on specific requirements (cf. Sect. 4). Further, if additional verification needs are identified in this phase, we could easily expand the verification report in an iteration.

Beside the pure verification of the safety requirements, the formal model can be used for a variety of other methods to aid in the assessment, e.g., Failure Analysis [12], Fault Tree Analysis [7,18] or Deductive Cause Consequence Analysis [13].

3 B: Semi-formal System Analysis

During this phase, the functional system architecture and representative system behavior descriptions are created from a subset of categorized requirements. The activities of this phase are depicted in Fig. 3.

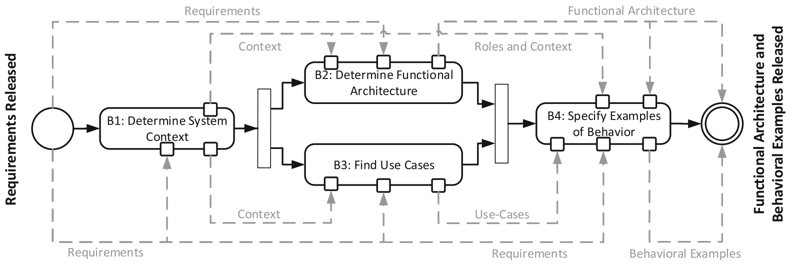


Fig. 3. The *Semi-Formal System Analysis* consists of five different phases. The goal of this phase is to derive a functional architecture and examples of the expected behavior of the system from the requirements.

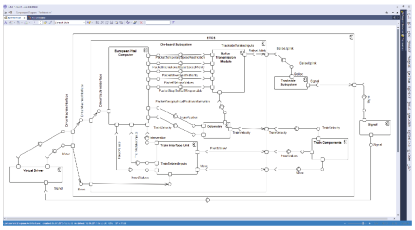
3.1 B1: Determine System Context

In this phase, we determine the border of the system. We define the system-to-be-developed, other external systems, and their roles that directly or indirectly interact with the system-to-be-developed. This determines the context of the specified system. Several external systems may be classified as the same role while other components directly represent their role. The goal is to achieve a consent with the involved stakeholders about the difference and similarities of external systems. This reassembles a typical engineering task - thus, we do not describe it in further detail. The context definition is used by the system architect to model the functional architecture of the system.

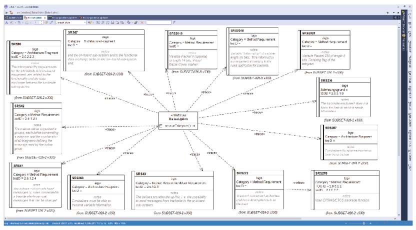
In the case study, we identified the border of the system as follows: The system-to-be-developed is the *European Vital Computer*. We identified different surrounding systems and roles, e.g., trackside subsystem, balise transmission unit, train interface unit, train components, and the driver.

3.2 B2: Determine Functional Architecture

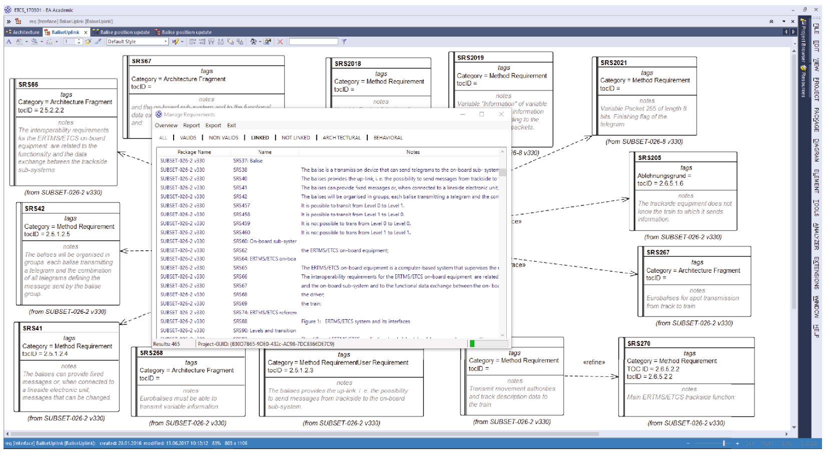
In this phase, we design the functional architecture by modeling system modules and their relations and interfaces. We will use the architecture to design a formal model and to implement the system in later phases.



(a) The architecture of our case study. Different components are assembled via ports, which implement interfaces.



(b) Requirement diagram for the interface *BaliseUplink*. Multiple requirements have been linked to the interface.



ETCS_170301 - EA Academic

req [Interface] BaliseUplink [BaliseUplink]

SRS66
tags
Category = Architecture Fragment
tocID = 2.5.2.2.2
notes
The interoperability requirements for the ERTMS/ETCS on-board equipment are related to the functionality and the data exchange between the trackside sub-systems
(from SUBSET-026-2 v330)

SRS42
tags
Category = Method Requirement
tocID = 2.5.1.2.5
notes
The balises will be organised in groups, each balise transmitting a telegram and the combination of all telegrams defining the message sent by the balise group.
(from SUBSET-026-2 v330)

SRS41
tags
Category = Method Requirement
tocID = 2.5.1.2.4
notes
The balises can provide fixed messages or, when connected to a lineside electronic unit, messages that can be changed.
(from SUBSET-026-2 v330)

SRS67
tags
Category = Architecture Fragment
tocID =

SRS2018
tags
Category = Method Requirement
tocID =

SRS2019
tags
Category = Method Requirement
tocID =
notes
Variable "Information" of variable

SRS2021
tags
Category = Method Requirement
tocID =
notes
Variable Packet 255 of length 8 bits. Finishing flag of the telegram
(from SUBSET-026-8 v330)

SRS205
tags
Ablehnungsgrund =
tocID = 2.6.5.1.6
notes
The trackside equipment does not know the train to which it sends information.
(from SUBSET-026-2 v330)

SRS267
tags
Category = Architecture Fragment
tocID =
notes
Eurobalises for spot transmission from track to train.
(from SUBSET-026-2 v330)

SRS268
tags
Category = Architecture Fragment
tocID =
notes
Eurobalises must be able to transmit variable information.
(from SUBSET-026-2 v330)

tags
Category = Method RequirementUser Requirement
tocID = 2.5.1.2.3
notes
The balises provides the up-link, i. e. the possibility to send messages from trackside to the on-board sub-system.
(from SUBSET-026-2 v330)

tags
Category = Method Requirement
tocID =
notes
Transmit movement authorities and track description data to the train
(from SUBSET-026-2 v330)

«refine»

SRS270
tags
Category = Method Requirement
TOC ID = 2.6.5.2.2
tocID = 2.6.5.2.2
notes
Main ERTMS/ETCS trackside function:
(from SUBSET-026-2 v330)

Manage Requirements

ALL | VALIDS | NON VALIDS | LINKED | NOT LINKED | ARCHITECTURAL | BEHAVIORAL

Package Name	Name	Notes
SUBSET-026-2 v330	SRS37: Balise	
SUBSET-026-2 v330	SRS38	The balise is a transmission device that can send telegrams to the on-board sub- system
SUBSET-026-2 v330	SRS40	The balises provides the up-link, i. e. the possibility to send messages from trackside to
SUBSET-026-2 v330	SRS41	The balises can provide fixed messages or, when connected to a lineside electronic unit,
SUBSET-026-2 v330	SRS42	The balises will be organised in groups, each balise transmitting a telegram and the com
SUBSET-026-2 v330	SRS457	It is possible to transit from Level 0 to Level 1.
SUBSET-026-2 v330	SRS458	It is possible to transit from Level 1 to Level 0.
SUBSET-026-2 v330	SRS459	It is not possible to trans from Level 0 to Level 0.
SUBSET-026-2 v330	SRS460	It is not possible to trans from Level 1 to Level 1.
SUBSET-026-2 v330	SRS60: On-board sub-syster	
SUBSET-026-2 v330	SRS62	the ERTMS/ETCS on-board equipment;
SUBSET-026-2 v330	SRS64: ERTMS/ETCS on-boa	
SUBSET-026-2 v330	SRS65	The ERTMS/ETCS on-board equipment is a computer-based system that supervises the
SUBSET-026-2 v330	SRS66	The interoperability requirements for the ERTMS/ETCS on-board equipment are related
SUBSET-026-2 v330	SRS67	and the on-board sub-system and to the functional data exchange between the on- boa
SUBSET-026-2 v330	SRS68	the driver;
SUBSET-026-2 v330	SRS69	the train;
SUBSET-026-2 v330	SRS74: ERTMS/ETCS referen	
SUBSET-026-2 v330	SRS88	Figure 1: ERTMS/ETCS system and its interfaces
SUBSET-026-2 v330	SRS90: Levels and transition	

Results: 465 | Project-GUID: {B30D7B65-9DE0-432c-AC98-7DC8366DE7C9}

req [Interface] BaliseUplink [BaliseUplink]: created: 28.01.2016 modified: 13.06.2017 10:12:12 83% 803 x 1106

(c) Requirement validation report view: different reports have been implemented. Based on the requirement categorization we can identify requirements correctly linked to semi-formal elements.

Fig. 4. An excerpt of the functional architecture. Figure 4a depicts an excerpt of the architecture of our case study. Figure 4b depicts a requirement diagram, demonstrating the traceability approach for a single interface. Figure 4c depicts the requirement validation view implemented as a plugin in Enterprise Architect.

We identify components, ports, and interfaces from the SRS, in specific from the subset of architecture, method and glossary requirements. We use the requirements according to their categorization: Glossary fragments, architecture and method requirements represent architectural statements in the requirement pattern [10] used in this paper. Components are derived from Glossary fragments. Relations of components (e. g., ports or assemblies) are identified via architecture requirements, whereas method requirements are used to derive method definitions for interfaces. A more detailed look on this is given in our previous work in [11].

The functional architecture of the ETCS model has been derived from 363 Requirements and consists of 31 components with 135 Ports as well as 23 interfaces with 161 methods and 136 corresponding call parameters. An excerpt is shown in Fig. 4a. We directly linked (via «trace» relations) the requirements and the semi-formal elements during this phase. This ensures traceability. Further, we set ourselves the following goal: Every semi-formal element shall be linked to at least one requirement.

During this phase, it is vital to monitor the traceability. Thus, we designed traceability reports: a straight forward report to identify the elements linked to requirements and those who are not. However, due to the usage of a requirement pattern, other reports are of more value: We design a report to identify the architectural elements, which have been linked to requirements. Further, we can identify the requirements, that *should* have been linked to some architectural element - but are not. We also designed these reports for other requirement categories, e. g., behavioral requirements.

These reports have been proven their value during the development: They enable us to detect which requirements still have to be processed and to evaluate the progress of the current phase. They also enable us to identify miss-categorized requirements or identify incorrect and inconsistent requirements due to the elements introduced during the modeling and then start a requirement change request (See Fig. 1). They further help to increase the acceptance of the model of others, who are not directly involved in the development, e. g., assessors, since the reports enable others to judge the requirement linkage.

3.3 B3: Find Use Cases

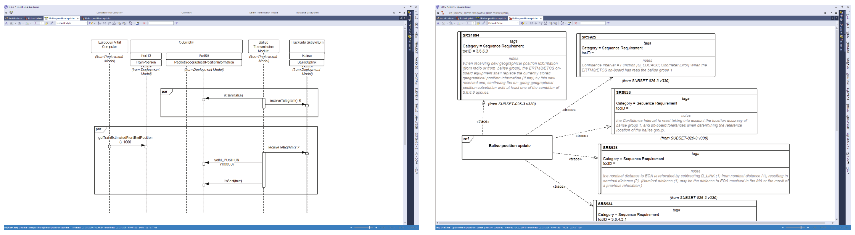
Besides the definition of the semi-formal architecture, we require a definition of behavior. Based on the context definition, the system analyst needs to define use cases. These use cases enable us to derive a top-level behavioral description of the SRS. Further, we require the analyst to link the use cases and requirements. After this phase, we refine the behavioral descriptions with the help of sequence diagrams.

Based on the previously established context definition, the use cases and the traceability reports for a subset of behavioral requirements (in specific user requirements) we can enhance the quality of our semi-formal model. If we identify use cases that are not traced to a requirement, we either identified creativity in the semi-formal model or identified an inconsistency in the requirements. Further, if we identify behavioral requirements that are not used to derive a use case, we identified a blind spot of the semi-formal model.

These use cases and our traceability reports help us to enforce a consistent, complete and correct semi-formal behavioral description. However, these use cases describe an overview of the behavior - we refine these descriptions with the help sequence diagrams in phase B4.

3.4 B4: Specify Expected Behavior

Use cases describe the basic behavior of the system. However, they are to imprecise to derive internal system behavior. Further, the behavioral description within the SRS is often incomplete: The SRS typically specifies the expected behavior. However, it does not describe the (mathematically) complete behavior. Hence, we use sequence diagrams to model the system's behavior: A set of sequence diagrams is modeled by domain experts from the set of behavioral requirements until the necessary behavior is described. The set of modeled sequences is used in later phases to demonstrate that the formal model contains the expected behavior of the semi-formal system model (*cf.* Sect. 4). We chose sequence diagrams instead of activity charts because they allow each to model one exemplary behavior trace of the system, whereas an activity diagram would imply that it defines the only paths the system model is allowed to represent. Every sequence diagram must be linked to a requirement. An example is depicted in Fig. 5. The traceability reports in combination with the requirement pattern enable us to detect incomplete elements and missing requirements.



(a) Example sequence from the case study: it describes the update of the position information via a balise.

(b) Requirement diagram to demonstrate the requirement linkage of the sequences depicted in Fig. 5a.

Fig. 5. Exemplary sequence diagram taken from the case study. Every sequence has to be traced to at least a single requirement as shown in Fig. 5b.

4 C: Formal Requirements Verification

This phase, the *Formal Requirements Verification*, covers the implementation of the formal model and the verification of the SRS with the help of model checking. It consists of five different phases (*cf.* Fig. 6).

4.1 C1: Design Formal Model

From the functional architecture, the formal model's architecture is generated: semi-formal elements are automatically translated into their formal representation. The basic idea is that every UML element translates into some formal element, such that the translated formal element maps to a specific semi-formal

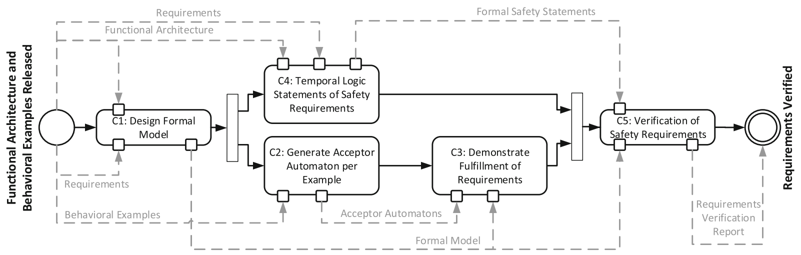
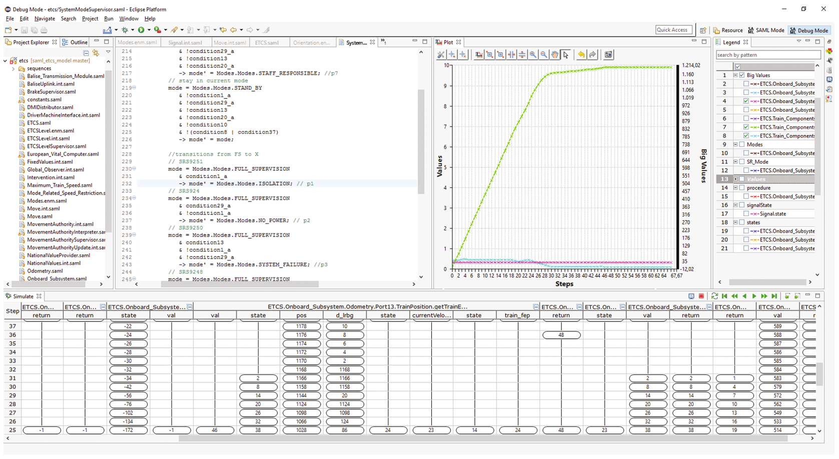

Functional Architecture and Behavioral Examples Released
Requirements
Functional Architecture
Formal Safety Statements
C1: Design Formal Model
C4: Temporal Logic Statements of Safety Requirements
C2: Generate Acceptor Automaton per Example
C3: Demonstrate Fulfillment of Requirements
C5: Verification of Safety Requirements
Requirements Verified
Requirements
Behavioral Examples
Acceptor Automatons
Formal Model
Requirements Verification Report

Fig. 6. The *Formal Requirements Verification* consists of five phases. The goal is to derive a formal model and verify safety requirements.

element. For example, the semi-formal architecture (e. g., a set of hierarchical components) is translated into a set of formal, hierarchical components. We refer the reader for further details on the transformation to our work in [11].

Given the formal representation, we can automatically preserve traceability: Every formal element is annotated with the unique ID of the semi-formal element which it represents. Further, we automatically translate the requirements linked to the specific semi-formal element. An example of the formal model is depicted in Fig. 7.

The task of the modeler is to implement the formal behavior according to the requirements. To do so, he has to formalize the behavior specified in the SRS. He has to rely on the architecture while implementing the automata. Further, he has to add a requirement link manually to the formal model, to preserve traceability.



```
          & !condition29_a
          & !condition13
          & !condition20_a
          -> mode' = Modes.Modes.STAFF_RESPONSIBLE; //p7
       // stay in current mode
       mode = Modes.Modes.STAND_BY
          & !condition1_a
          & !condition29_a
          & !condition13
          & !condition20_a
          & !condition10
          & !(condition8 | condition37)
          -> mode' = mode;

       //transitions from FS to X
       // SRS9251
       mode = Modes.Modes.FULL_SUPERVISION
          & condition1_a
          -> mode' = Modes.Modes.ISOLATION; // p1
       // SRS924
       mode = Modes.Modes.FULL_SUPERVISION
          & condition29_a
          & !condition1_a
          -> mode' = Modes.Modes.NO_POWER; // p2
       // SRS9250
       mode = Modes.Modes.FULL_SUPERVISION
          & condition13
          & !condition1_a
          & !condition29_a
          -> mode' = Modes.Modes.SYSTEM_FAILURE; //p3
       // SRS9248
       mode = Modes.Modes.FULL_SUPERVISION
```

Step	ETCS.On...	ETCS.On...	ETCS.Onboard_Subsyste...		ETCS.Onboard_Subsystem.Odometry.Port13.TrainPosition.getTrainE...								ETCS.On...	ETCS.On...	ETCS.Onboard_Subsyste...		ETCS.On...	ETCS.On...
	return	return	state	val	val	state	pos	d_lrbg	state	currentVelo...	state	train_fep	return	state	val	return	return	val
37			-22				1178	10										589
36			-24				1176	8					48					588
35			-26				1174	6										587
34			-28				1172	4										586
33			-30				1170	2										585
32			-32				1168	1168										584
31			-34			2	1166	1166							2	2	1	583
30			-42			8	1158	1158							8	8	4	579
29			-56			14	1144	20							14	14	7	572
28			-76			20	1124	1124							20	20	10	562
27			-102			26	1098	1098							26	26	13	549
26			-134			32	1066	124							32	32	16	533
25	-1	-1	-172	-1	46	38	1028	86	24	23	14	24	48	23	38	38	19	514

Fig. 7. A screenshot of the VECS debugger with the ETCS case study model. Here, the fulfilling of a sequence has been analyzed. As a result, a representative model path is given that can be used to check the validity of the sequence and for documenting the validity of the formal model for the assessment.

Based on these links, it is possible to generate requirement traceability reports to enhance the overall quality as described in Sect. 3.2.

For the ETCS case study, we implemented 53 state variables with 444 state transition rules so far (resulting in a state space of 2×10^{25} possible states[3]) from 1.511 requirements. In average, we modeled about 300 requirements per working day. Although the formal model is implemented to best of the modelers' knowledge, it might contain faults. The following phases aim at validating the formal implementation.

4.2 C2: Generate Acceptor Automaton per Example and C3: Demonstrate Fulfillment of Requirements

The goal of these phases is to show that the intended behavior is implemented in the formal model. We make use of sequence diagrams (cf. Sect. 3.4) to demonstrate the complete behavior implementation.

The formal model is an automaton. Thus its semantic is the set of all independent traces which are generated by executing it. A sequence diagram represents a single execution trace [17]. We say that the formal model covers a sequence if we can show that it is a sub-trace of at least one trace within the semantics of the formal model. To verify this, the sequence is translated into an acceptor automaton relying on the same architecture as the UML model, such that it moves to the next state every time a message occurred as described in the sequence.

We verify that the last state is reachable with the help of model checking [4]. To verify the reachability retractable, we provoke a witness. A witness is an example path demonstrating the reachability of the acceptor automaton's last state. Instead of checking whether the acceptor automaton's last state is reachable, we verify whether the last state of the acceptor automaton is not reachable. If it is not reachable, the model checker returns true, and we can check, e.g., if the previous state is reachable. But, if the last state is reachable, the model checker returns false and generates a path to the desired state as a counterexample. This example path can be checked by an external expert and supports the reliance on the formal model as it retraceable shows the existence of the sequence in the semantic of the formal model.

Up to now, the demonstration of the fulfillment of requirements for our case study is not 100% completed, because we have not demonstrated that *every* sequence is present in the formal model. However, we were able to demonstrate that *some* sequences are present. Based on the first results, we detected faults and inconsistencies in the formal model. Further, we were able to increase the reliance of others on our model: especially others without a strong formal background gained trust in the formal model.

[3] Worst case approximation by multiplying all possible state variable values.

4.3 C4: Temporal Logic Statements of Safety Requirements and C5: Verification of Safety Requirements

The safety requirements can be verified as soon as they are implemented. To do so, the formal modeler translates the natural language safety requirements into the used formal language (cf. Fig. 6). The formal language corresponds to preferred verification engine, i.e., the model checker. In most cases, this is either Computation Tree Logic or Linear-time Temporal Logic [4].

The output the model checker generated during the verification contains the result (cf. Fig. 6 *Verification of Safety Requirements*). These results demonstrate whether the specification fulfills the safety requirements. If the formal model fulfills the safety requirements, the system requirement specification covers the desired safety specifications. This holds if we applied the proposed process and demonstrated completeness and consistency.

5 Related Work

Hallerstede et al. [14] transformed requirements directly into a formal model. They offer tool support through Rodin and ProR to trace requirements. However, the direct transformation, in contrast to our approach, seems much more difficult to perform as it requires extensive expert knowledge in the domain of the system and the used formal verification technique.

Aceituna and Do propose an approach to find errors in specifications [1]. Our goals differ slightly, as the authors try to expose possible, unintended, off-nominal behavior. In contrast to that, our approach has a broader scope: verifying safety specifications, whereas off-nominal behavior can be detected as a side product during the verification if it opposes the safety goals. Aceituna et al. translate natural language requirements into causal components [1,2]. The authors propose to expand the transition rules of the system by modeling explicit rules for the entire state space. Although the authors display the feasibility of the proposed approach, their case studies are rather small. For large system requirement specifications, this approach seems to unfeasible, because of the need to expand the transition rules. In a previous work [2], the authors demonstrated how causal components can directly be mapped into SMV. However, the approach lacks some behavior validation mechanism to ensure wanted behavior to be present in the model.

An approach enforcing a correct behavior translation through automatic validation as been proposed by Soliman et al. [19]. A test case generator is used to perform the automatic validation of the transformation process based on scenarios. The generated test cases are integrated into the safety application timed-automata network and simulated automatically with UPPAAL. Finally, the output variables have to be compared. The authors did not specify if the comparison can be done automatically and did not ensure traceability.

The COMPASS toolset [6] is quite similar to our approach: A formal model is used for a variety of model-based verifications. However, the toolset requires a formal model; it does not address its derivation from a SRS. Recently, the toolset was extended with a specific requirement pattern, originated in the aerospace domain, to structure the derivation [5]. However, the approach lacks a semi-formal phase which increases the understandability of the resulting formal model.

6 Conclusions and Future Work

In this paper, we presented our approach for integrating formal verification techniques into the safety lifecycle of critical systems. This includes a well-defined and structured process which supports the creation of a formal model from a given set of system requirements. For increasing the acceptance, in particular, during the assessment, of the verification artifacts and the applicability of the process, we integrated structural UML models in a semi-formal step. This supports the comprehension of the model and reliability of all involved in the correctness of the formal model.

Together with our project partners, we demonstrated the applicability of our approach on an industrial-size case study of the European Train Control System. In this context, we processed about 4.200 requirements from functional description (SUBSET-026) of the ETCS, containing more than 33.000 requirements in total.

From this, we built a verifiable formal model, which were also accepted by the project partner as a reliable representation of the system specification. In particular, this resulted from the rigorous implementation of the traceability and the use of UML sequence charts as a measure of the model's validity. Further, this was supported by the implementation of the semi-formal UML phase introducing the representation of the formal model's structure in a widely accepted and known representation.

Also, another important step towards the integration of the process is its implementation within our formal modeling and verification tool chain VECS, offering an interface to the widely used requirement tool Rational Doors and the UML modeling tool Enterprise Architect. Whereby, the traceability and, moreover, the comprehensibility of the modeling and verification results got improved.

Raising the quality of the specification on a trustworthy level should always be one central demand. However, if the formal model has been build once it should also be used for further safety measurements. Therefore, we plan to integrate several further model-based analysis methods, e.g., Fault Tree Analysis in the next step. Besides this, we are also working on the enabling correlation and trace refinement measures between the formal model and the implemented program code, to open the process integration not only for new developments but also for extensions of already existing systems.

Acknowledgments. The work presented in this paper is funded by the German Ministry of Education and Science (BMBF) in the VIP-MoBaSA project (project-Nr. 16V0360).

References

1. Aceituna, D., Do, H.: Exposing the susceptibility of off-nominal behaviors in reactive system requirements. In: RE, pp. 136–145 (2015). doi:10.1109/RE.2015.7320416
2. Aceituna, D., Do, H., Srinivasan, S.: A systematic approach to transforming system requirements into model checking specifications. In: ICSE, pp. 165–174 (2014). doi:10.1145/2591062.2591183
3. Ammann, P.E., Black, P.E., Majurski, W.: Using model checking to generate tests from specifications. In: Proceedings of the Second International Conference on Formal Engineering Methods, pp. 46–54. IEEE (1998). doi:10.1007/3-540-48166-4_10
4. Baier, C., Katoen, J.P., Larsen, K.G.: Principles of Model Checking. MIT Press, Cambridge (2008). ISBN: 9780262026499
5. Bos, V., Bruintjes, H., Tonetta, S.: Catalogue of system and software properties. In: Skavhaug, A., Guiochet, J., Bitsch, F. (eds.) SAFECOMP 2016. LNCS, vol. 9922, pp. 88–101. Springer, Cham (2016). doi:10.1007/978-3-319-45477-1_8
6. Bozzano, M., Cimatti, A., Katoen, J.P., Nguyen, V.Y., Noll, T., Roveri, M.: The COMPASS approach: correctness, modelling and performability of aerospace systems. In: Buth, B., Rabe, G., Seyfarth, T. (eds.) SAFECOMP 2009. LNCS, vol. 5775, pp. 173–186. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04468-7_15
7. Bozzano, M., Cimatti, A., Tapparo, F.: Symbolic fault tree analysis for reactive systems. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 162–176. Springer, Heidelberg (2007). doi:10.1007/978-3-540-75596-8_13
8. Cimatti, A., Roveri, M., Susi, A., Tonetta, S.: From informal requirements to property-driven formal validation. In: Cofer, D., Fantechi, A. (eds.) FMICS 2008. LNCS, vol. 5596, pp. 166–181. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03240-0_15
9. EN 50128: Railway applications-communication, signaling and processing systems-software for railway control and protection systems (2011)
10. Filax, M., Gonschorek, T., Lipaczewski, M., Ortmeier, F.: On traceability of informal specifications for model-based verification. In: IMBSA: Short & Tutorial Proceedings, pp. 11–18. OvGU Magdeburg (2014)
11. Filax, M., Gonschorek, T., Ortmeier, F.: Correct formalization of requirement specifications: a v-model for building formal models. In: Lecomte, T., Pinger, R., Romanovsky, A. (eds.) RSSR 2016. LNCS, vol. 9707, pp. 106–122. Springer, Cham (2016). doi:10.1007/978-3-319-33951-1_8
12. Ge, X., Paige, R.F., McDermid, J.A.: Analysing system failure behaviours with PRISM. In: SSIRI-C, pp. 130–136 (2010). doi:10.1109/SSIRI-C.2010.32
13. Güdemann, M., Ortmeier, F., Reif, W.: Using deductive cause-consequence analysis (DCCA) with SCADE. In: Saglietti, F., Oster, N. (eds.) SAFECOMP 2007. LNCS, vol. 4680, pp. 465–478. Springer, Heidelberg (2007). doi:10.1007/978-3-540-75101-4_44
14. Hallerstede, S., Jastram, M., Ladenberger, L.: A method and tool for tracing requirements into specifications. Sci. Comput. Program. **82**, 2–21 (2014). doi:10.1016/j.scico.2013.03.008
15. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems (2005)
16. ISO 26262: Road Vehicles-Functional Safety (2009)
17. OMG UML: Unified modeling language, superstructure (2011)

18. Ortmeier, F., Schellhorn, G.: Formal fault tree analysis-practical experiences. Electron. Not. Theoret. Comput. Sci. **185**, 139–151 (2007). doi:10.1016/j.entcs.2007.05.034
19. Soliman, D., Frey, G., Thramboulidis, K.: On formal verification of function block applications in safety-related software development. IFAC **46**(22), 109–114 (2013). doi:10.3182/20130904-3-UK-4041.00015
20. Withall, S.: Software Requirement Patterns (Developer Best Practices). Microsoft Press (2007). ISBN: 9780735623989