On Traceability of Informal Specifications for Model-Based Verification

Marco Filax, Tim Gonschorek, Michael Lipaczewski, and Frank Ortmeier

Chair of Software Engineering,
Otto-von-Guericke University of Magdeburg,
Germany
https://cse.cs.ovgu.de

Abstract. Safety critical systems are often specified by a set of informal requirements. The fulfillment of those requirements has, in general, to be verified through external assessors. In most cases models (i.e. UML diagrams) are used to communicate system architectures and decide whether a given system meets its requirements. Even though, there already exist multiple approaches to verify safety critical systems and assessors would benefit from the usage of model-based system verifications, they are not commonly used in industry. We propose a traceable modeling approach for verifying safety critical systems specified with a set of informal unstructured requirements. The proposed process is divided into three phases: the analysis of the given set of informal requirements, semi-formalization and formalization.

Keywords: Safety Assessment, Formal Verification, Specifications, Requirement Engineering

1 Introduction

Systems requirement engineering is important in industrial projects [1, 11]. Utilizing semi-formal modeling techniques, such as UML (Unified Modeling Language) [15] are common practice [12, 19]. As specifications are typically written in natural language, the processing and tracing of requirements commonly must be done manually by domain experts. During this process, traceability is important as it is defined to be the ability of following a requirement from its origins to its development, specification and use [7]. Thus, traceability can become critical to a projects success [20].

When specifying safety critical systems, there often exists the need to verify the correct behavior of the system-to-be. Model-based safety analysis can help detecting system failures by utilizing an abstract formal model. This emphasizes the observation of several research projects trying to apply formal model-based analysis to verify safety critical systems [13, 14, 6, 5]. These approaches are not widely accepted in industry, as requirements have to be processed by domain experts, who are commonly not familiar with formal methods [4].

2 The Verification Process

The proposed process is outlined in Figure 1. We divide the verification process into three phases, consisting of the analysis of informal requirements, the semi-formalization of requirements through an industrial approved subset of UML and the actual formalization and verification phase.

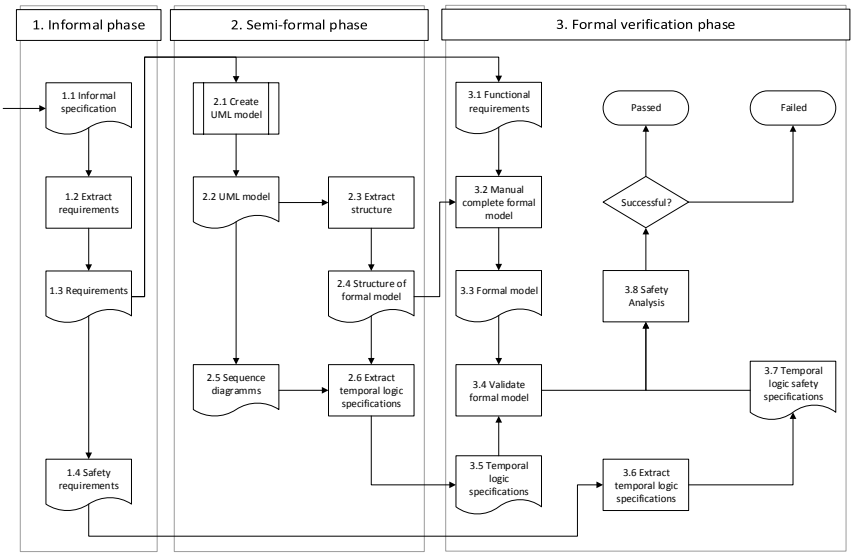

Fig. 1. The proposed process with its three phases.

2.1 Analysis of Informal Requirements

As Ryan [18] already predicts decades ago, current research has shown that system behavior specifications are still written in natural language [10]. Although there exist numerous approaches that try to extract requirements automatically [6, 5], industrial projects have shown that these approaches cannot be applied productively, because they share the need of writing specifications with structured natural language. Because of that, it is still common practice to define different requirement categories and classify text fragments. We adapt the methodology of Cimatti et al. [4] and extend it with additional categories.

We distinguish eight different categories listed in Table 1. According to the appropriate fragment condition, natural language specification fragments have to be manually clustered into the corresponding category. As there already exists elaborate tool support for categorizing requirement specification documents (i.e. DOORS) this issue is not further addressed in this paper.

Fragment category	Fragment condition
Glossary fragment	Does the text fragment define a specific concept of the domain?
Architecture fragment	Does the text fragment introduce some system's modules and describe how they interact?
Functional requirement	Does the text fragment describe the steps a particular module performs or the states where a module might be in?
Communication requirement	Does the text fragment describe messages modules exchange?
Property requirement	Does the text fragment describe expected properties of the domain or constraints of the system-to-be?
User requirement	Does the text fragment describe actions or constraints which have to be considered, satisfied or performed by the user?
Safety requirement	Does the text fragment describe necessary safety constraints?
Annotation	Is the text fragment a note in the specifications that does not add any information about the ontology or the behavior of the specified system?

Table 1. Categories to classify text fragments, adapted from Cimatti et al. [4]

2.2 Semi-Formalization through UML

The semi-formal verification phase comprises the creation and validation of an UML model. We propose to create different structural and behavioral UML artifacts. Additionally, temporal-logic specifications can be extracted automatically from behavioral diagrams while structural diagrams can be transformed into a partial formal representation.

Creating the UML Model (see Figure 1, step 2.1) covers the process of creating the required diagrams. Depending on the specific text fragment category, defined in Table 1, the fragment has to be translated into the according UML artifact. Figure 2 is a recommendation for translating text fragments into semi-formal representations and may be adapted.

Component diagrams are structural diagrams visualizing the overall system architecture. As glossary fragments are defined to represent specific domain concepts, they can be exploited to extract naming conventions for possible components. Since architecture fragments contain additional information about components of the system-to-be, they can be modeled in UML. The creation of component diagrams supports the further development of the UML model, as they increase the clarity of the system.

Use case diagrams are extracted using the pre-processed user requirements. Extracting use cases can help gathering the required sequences that have to be modeled. For example, names of activities, extracted from text fragments give possible names for sequence charts.

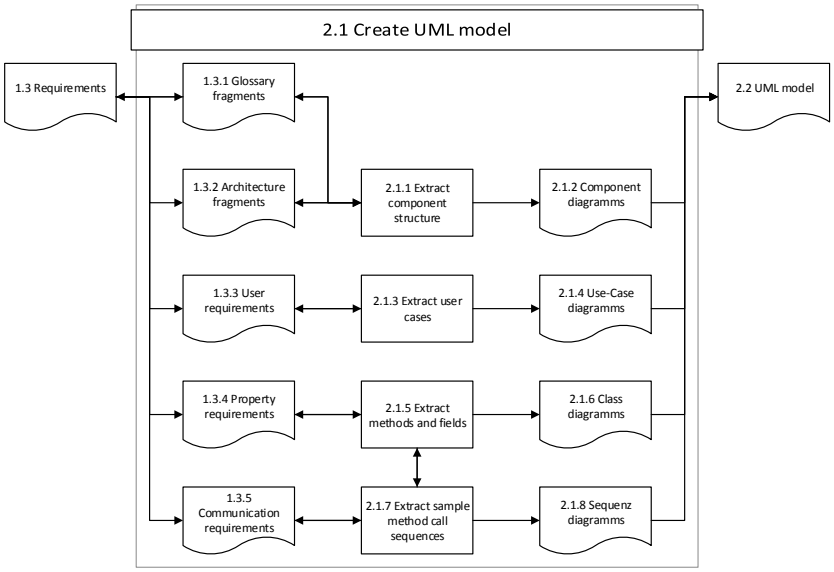

Fig. 2. Different sub-processes of the UML-model creation (see Figure 1, step 2.1)

Class diagrams are developed as denoted in Figure 2. During this step property requirement fragments are used in order to extract classes, attributes and function names. In the following steps, class artifacts are translated into a partial formal model. Thus, they are vital to maintain traceability in the overall process, as they represent the link of the formal model to the informal specification.

Sequence diagrams are behavioral diagrams that illustrate the operation of the system-to-be. As communication requirement fragments are defined to contain messages that modules can exchange, it is possible to extract sample method call sequences. Therefore, communication requirement fragments can be utilized to extract sequence chart diagrams.

The generation of multiple sequence diagrams is an important step, as sequences can be transformed to temporal-logic specifications. Thus, they are vital to prove the correlation of the formal and semi-formal model.

The creation of individual UML artifacts may and should be done in a parallel way. This results in an incremental refinement of the requirement fragments or diagrams, in general. Change management then plays an important role. It may also be supported by this three-phased process, but is not discussed in this paper. To maintain traceability, the creation and adaption of UML objects has to rely on the pre-processed requirement fragments. These dependencies have to be manually denoted through the modeler, i.e., using SysML [16].

Extracting the structure of the formal model represents the process of transforming an UML model to a partial formal model (see Figure 1, step 2.3). Structural UML diagrams yield enough information to generate source code headers. Therefore, names of formal components and state names can be extracted from UML components and classes [13, 14].

Every generated formal element has to be linked, to either an UML artifact or requirement. It is clearly advised to extract the structure of the formal model automatically. This also enables the automated generation of traceability links. The usage of the generated partial formal model in combination with automatic extracted temporal-logic specifications enables the automatic validation of the complete model.

Extracting temporal-logic specifications is depicted in Figure 1, step 2.6. Using the structure of a formal model, including components, states and formulas, facilitates the extraction of temporal-logic specifications [14]. Sequence diagrams represent exemplary interactions of given instances utilizing an array of messages. We propose to exploit formula-like structures to represent these sequences in a formal way.

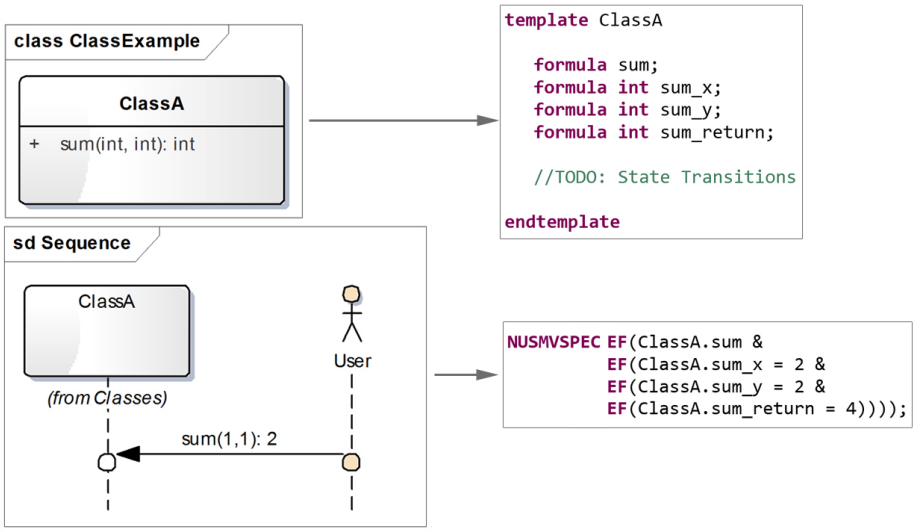

Fig. 3. Example extraction of the partial formal model and a sample specification.

Figure 3 depicts an example of the proposed process. As shown in the image, the class *classA* is translated to a partial formal model. Using the generated structure, it is possible to extract temporal specifications automatically, i.e., by exploiting formula-like structures in SAML [8]. As formulas are defined as stateless expressions, they can refer to any state or combination of states, which

can be derived to forward information instantaneously while using discrete finite automata. Thus, we propose to utilize at least two formulas to represent a method signatures and the return type. Moreover, every function parameter can be represented by another formula.

Transforming glossary and architecture fragments as well as user, property and behavioral requirements to an UML model, the subsequent the extraction of the formal structure and temporal logic specifications represents the overall semi-formal phase of the proposed process. The gathered results form the basis for the final formal verification phase.

2.3 Formal Verification Phase

The formal verification phase consists of the finalization and validation of the formal model. Further, the sub-processes depicted in Figure 1 are explained in detail.

The manual completion of the formal model represents the most important sub-process of the formal verification phase. Every functional requirement fragment has to be translated into a set of state transitions. Thus, it is necessary that functional requirement fragments and state transitions are linked. In its simplest form, this might be done by structured comments. Moreover, tool support in forms of context menus is possible.

Validating the formal model is important, as the consistency of the informal specification and the formal model has to be shown. During the process, sequence diagrams were developed and translated into temporal-logic specifications. Therefore, the formal model has to satisfy every temporal-logic sequence representation in order to be valid. Otherwise, the correct transformation of functional fragments to state transitions has to be reviewed. If the validation is successful, the formal model can be used to perform the actual safety analysis.

Extracting temporal-logic safety specifications is a sub-process using safety requirement fragments depicted in Table 1. Temporal-logic specifications are required in order to perform the formal safety analysis. As safety requirement fragments are commonly defined to describe general constraints, they can only be modeled using domain knowledge. Therefore, it is essential to document this process-step extensively. The construction and documentation of temporal-logic specifications is followed by the actual safety analysis.

The safety analysis of an informal specified system represents the main goal of the proposed process, thus, it also represents the final step. During this step, a model checker verifies if the formal model meets every temporal-logic safety specification. There already exist multiple model checkers like PRISM, NuSMV, MRMC or Spin, who are capable of verifying a formal model. Depending on the features supported, it is necessary to choose the appropriate solution.

3 Related Work

Central to this paper is the categorization approach of Cimatti et al. [3, 4, 2]. We adopted the methodology to categorize requirement fragments, without relying on the need of writing controlled natural language.

Another example for formal verification while preserving traceability is Rodin in combination with ProR [9]. Utilizing a different requirement fragment categorization, Hallerstede et al. transform requirements directly to a formal representation in EventB. This direct transformation in contrast to our approach seems much more difficult to perform.

The authors in this paper [17] proposed a approach utilizing state charts and internal block diagrams to verify requirements using UPAAL. Modeling complex system entirely through state charts and block diagrams seems to be a challenging task.

4 Conclusions and Further Work

In this paper, we proposed a three phase methodology to generate a formal representation of an informal specification. Using the formal model it is possible to verify whether the system-to-be meets its safety requirements. The declined key goal of the proposed process is the transformation in a traceable matter. Therefore, we exploited an industrial approved subset of behavioral and structural UML diagrams. To generate the semi-formal artifacts we use categorized requirement fragments. Structural UML artifacts are used to automatically generate a partial formal model. Behavioral artifacts are translated to temporal-logic specifications, in order to verify the correspondence of the semi-formal model and the manually completed formal representation. After that, the formal model is validated and can be verified.

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. B.W. Boehm. Software risk management: principles and practices. *Software, IEEE*, 8(1):32–41, 1991.
2. A. Chiappini, A. Cimatti, L. Macchi, O. Rebollo, M. Roveri, A. Susi, S. Tonetta, and B. Vittorini. Formalization and validation of a subset of the european train control system. In *Proceedings of the International Conference on Software Engineering ()*, volume 2, pages 109–118, 2010.
3. A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. Formalization and validation of safety-critical requirements. In *FMA*, pages 68–75, 2009.

4. A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. From informal requirements to property-driven formal validation. In *Formal Methods for Industrial Critical Systems*, volume 5596 of *Lecture Notes in Computer Science*, pages 166–181. Springer Berlin / Heidelberg, 2009.
5. D. K. Deeptimahanti and R. Sanyal. Semi-automatic generation of uml models from natural language requirements. In *Proceedings of the India Software Engineering Conference (ISEC 2011)*, pages 165–174. ACM, 2011.
6. D.A. Ferreira and A.M.R. Silva. Survey on system behavior specification for extending projectit-rsl. In *Proceedings of the Quality of Information and Communications Technology (QUATIC 2010)*, volume 7, pages 210–215. IEEE Computer Society, Sept 2010.
7. O.C.Z. Gotel and A.C.W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the international conference on Requirements Engineering*, pages 94–101. IEEE Computer Society, 1994.
8. Matthias Güdemann. *Qualitative and Quantitative Formal Model-Based Safety Analysis*. PhD thesis, Otto-von-Guericke-Universitä Magdeburg, 2011.
9. S. Hallerstede, M. Jastram, and L. Ladenberger. A method and tool for tracing requirements into specifications. *Science of Computer Programming*, 82(0):2 – 21, 2014. Special Issue on Automated Verification of Critical Systems (AVoCS11).
10. S. Hansen, N. Berente, and K. Lyytinen. Requirements in the 21st century: Current practice and emerging trends. In *Design Requirements Engineering: A Ten-Year Perspective*, volume 14 of *Lecture Notes in Business Information Processing*, pages 44–87. Springer Berlin / Heidelberg, 2009.
11. I. Jacobson, G. Booch, J. Rumbaugh, and G. Booch. *The unified software development process*. Addison-Wesley Reading, 1999.
12. P. Kruchten. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.
13. L. Lavazza, G. Quaroni, and M. Venturelli. Combining uml and formal notations for modelling real-time systems. In *Proceedings of the European Software Engineering Conference Held Jointly with International Symposium on Foundations of Software Engineering*, ESEC/FSE-9, pages 196–206. ACM, 2001.
14. W.E. McUmber and B.H.C. Cheng. A general framework for formalizing uml with formal languages. In *Proceedings of the International Conference on Software Engineering (ICSE 2001)*, pages 433–442. IEEE Computer Society, 2001.
15. OMG. *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1*. Object Management Group, August 2011.
16. OMG. *OMG Systems Modeling Language, Version 1.3*. Object Management Group, 2012.
17. Jean-François Pétin, Dominique Evrot, Gérard Morel, Pascal Lamy, et al. Combining sysml and formal methods for safety requirements verification. In *Proceedings of the 22nd International Conference on Software & Systems Engineering and their Applications (ICSSEA 10)*, 2010.
18. K. Ryan. The role of natural language in requirements engineering. In *Proceedings of the International Symposium on Requirements Engineering*, pages 240–242. IEEE Computer Society, 1993.
19. I. Vessey and A.P. Sravanapudi. Case tools as collaborative support technologies. *Communications of the ACM*, 38(1):83–95, 1995.
20. R. Watkins and M. Neal. Why and how of requirements tracing. *Software, IEEE*, 11(4):104–106, July 1994.