

ANALYSIS OF COEXISTING GRAPHICAL AND TEXTUAL  
REPRESENTATIONS OF REQUIREMENTS BASED ON ACTIVITY  
DIAGRAMS AND STRUCTURED TEXT

vorgelegt von  
Master of Science  
Martin Beckmann  
geb. in Berlin

von der Fakultät IV - Elektrotechnik und Informatik  
der Technischen Universität Berlin  
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften  
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. habil. Odej Kao  
Gutachter: Prof. Dr. Andreas Vogelsang  
Gutachter: Prof. Dr. Jan Mendling  
Gutachter: Prof. Dr. Yasmina Bock

Tag der wissenschaftlichen Aussprache: 22. Januar 2019

Berlin 2019



## ABSTRACT

---

Due to increasing complexity and the tendency towards more customized products, the development of modern technical systems demands more and more effort. In an attempt to reduce the effort required, engineering models are being adopted to a greater extent in all phases of development. Among the affected development phases is requirements engineering. As a result, engineering models, which almost always take the form of graphical notation, have become more common in a setting where text-based requirements specifications have been historically predominant. These graphical models are not a replacement for text-based specifications, but supplement them instead. While these two forms of representation may be complementary, oftentimes they are strongly related or even express the same content. Hence, the graphical and textual representations coexist.

The use of coexisting graphical and textual representations for specification purposes has shown promising results. However, these results have been obtained in a research context and are at best evaluated in practice. In practice however, such approaches are tailored to domain- or even project-specific needs. It is unclear whether comparable approaches implemented by practitioners independently achieve the same promising results as those obtained in research contexts.

In this thesis, we address this gap by examining a case where an approach using coexisting graphical and textual representations is realized and productively used by practitioners. The specific realization of this approach employs *UML2 Activity Diagrams* as graphical representation and hierarchically structured text as textual representation to describe the functions of a system. From the results of this examination, we conclude that, for the most part, the practitioners perceive the benefits of this approach as predicted by research. Still, the application of such an approach in practice results in a number of pitfalls. Inconsistencies between the two representations, as well as other quality issues, constitute the biggest impact on the practical usability of this approach. Over 70% of the functions are affected by deficiencies that are considered major issues by users. As these deficiencies represent a sizable negative impact on the applicability of the approach, a method is developed in this thesis that prevents the occurrences of inconsistencies and other quality issues.

Overall, this thesis contributes to the body of knowledge of approaches that use an additional textual representation with graphical models. This knowledge may be employed to design new approaches or improve existing ones in order to increase acceptance among practitioners.

## ZUSAMMENFASSUNG

---

Aufgrund steigender Komplexität und der zunehmenden Tendenz zu individuell angepassten Produkten, erfordert die Entwicklung moderner technischer Systeme immer mehr Aufwand. Um diesen Herausforderungen beizukommen, werden zunehmend Modelle in allen Phasen der Entwicklung eingesetzt. Damit einhergehend ist auch das Anforderungsmanagement betroffen. Als Folge dessen findet der Einsatz von Modellen, welche meist in grafischer Notation vorliegen, zur Entwicklung mehr Verbreitung. Dies geschieht in einem Umfeld, in welchem aus historischer Sicht vor allem textbasierte Spezifikationen vorherrschend sind. Nichtsdestotrotz ersetzen die grafischen Modelle nicht die textuellen Spezifikationen, sondern erweitern diese. Dabei können diese verschiedenen Formen der Repräsentation einander ergänzen, oft sind diese jedoch eng miteinander verzahnt oder drücken sogar den gleichen Inhalt aus. Entsprechend liegt eine Koexistenz beider Repräsentationen vor.

Die Verwendung von koexistierenden grafischen und textuellen Repräsentation zum Zwecke der Spezifikation hat dabei vielversprechende Ergebnisse gezeigt. Jedoch stammen diese Ergebnisse lediglich aus einem wissenschaftlichen Umfeld und sind im besten Fall in einem industriellen Umfeld evaluiert worden. In der Praxis werden derartige Ansätze allerdings an die Domäne oder gar an projektspezifische Anforderungen individuell angepasst. Es bleibt daher ein offener Punkt, ob vergleichbare Ansätze, welche von Anwendern selbst implementiert werden, die gleichen Ergebnisse erzielen.

Solch eine Gegebenheit wird in dieser Arbeit untersucht. Das heißt, es wird ein Fall eines solchen Spezifikationsansatzes betrachtet, welcher koexistierende grafische und textuelle Repräsentation verwendet und von den Anwendern selbst umgesetzt wurde und produktiv genutzt wird. Konkret wird ein Ansatz betrachtet, welcher *UML2 Aktivitätsdiagramme* als die grafischen Modelle und hierarchisch strukturierten Text als die textuelle Repräsentation zur Beschreibung von Funktionen eines Systems verwendet. Aus den Ergebnissen der Betrachtung ergibt sich, dass die Anwender zum Großteil den von der Wissenschaft prognostizierten Nutzen ebenfalls wahrnehmen. Trotz dieser positiven Wahrnehmung, ergeben sich in der industriellen Anwendung eines solchen Ansatzes eine Reihe von Schwierigkeiten. Den größten Einfluss auf die praktische Verwendung stellen Inkonsistenzen zwischen den Repräsentation und andere Qualitätseinschränkungen dar. In dem untersuchten System sind 70% aller Funktionen von Defekten betroffen, welche die Anwender als große Probleme wahrnehmen. Da diese Defekte folglich eine beträchtliche negative Beeinträchtigung für die Anwendbarkeit des Ansatzes darstellen, wird in

dieser Arbeit eine Methode entwickelt, welche das Auftreten der Inkonsistenzen und anderen Qualitätseinschränkungen verhindert.

Insgesamt trägt diese Arbeit dazu bei, das Wissen zu Ansätzen, welche eine zusätzliche textuelle Repräsentation zu grafischen Modellen verwenden, zu konkretisieren und zu erweitern. Dies kann wiederum genutzt werden, um neue Ansätze zu kreieren oder existierende zu verbessern und somit die Akzeptanz von Nutzern zu steigern.



## PUBLICATIONS

---

The following publications are part of this thesis:

- [1] Martin Beckmann, Thomas Karbe, and Andreas Vogelsang. "Information Extraction from High-Level Activity Diagrams to Support Development Tasks." In: *International Conference on Model-Driven Engineering and Software Development* (2018).
- [2] Martin Beckmann, Vanessa N. Michalke, Andreas Vogelsang, and Aaron Schlutter. "Removal of Redundant Elements within UML Activity Diagrams." In: *International Conference on Model Driven Engineering Languages and Systems* (2017).
- [3] Martin Beckmann, Christian Reuter, and Andreas Vogelsang. "Coexisting Graphical and Structured-Textual Representations of Requirements: Insights and Suggestions." In: *International Working Conference on Requirements Engineering: Foundation for Software Quality* (2018).
- [4] Martin Beckmann, Andreas Vogelsang, and Christian Reuter. "A Case Study on a Specification Approach using Activity Diagrams in Requirements Documents." In: *International Requirements Engineering Conference* (2017).

Further ideas and figures have appeared previously in the following publications:

- [1] Martin Beckmann and Aaron Schlutter. "Automatische Duplikateliminierung in Aktivitätsdiagrammen von Fahrzeugfunktionen." In: *Workshop Automotive Software Engineering* (2016).
- [2] Martin Beckmann and Andreas Vogelsang. "Evaluation of a Specification Approach for Vehicle Functions using Activity Diagrams in Requirements Documents." In: *Workshop Automotive Software Engineering* (2017).
- [3] Martin Beckmann and Andreas Vogelsang. "What is a Good Textual Representation of Activity Diagrams in Requirements Documents?" In: *International Model-Driven Requirements Engineering Workshop* (2017).





# CONTENTS

---

<b>I</b>	<b>REQUIREMENTS SPECIFICATION WITH GRAPHICAL MODELS AND TEXT</b>	
1	INTRODUCTION	3
1.1	Motivation . . . . .	3
1.2	Context: Requirements Engineering in the Development of Automotive Systems . . . . .	5
1.3	Problem Statement . . . . .	6
1.4	Contributions of this Thesis . . . . .	7
1.5	Methodological Background . . . . .	11
1.6	Outline . . . . .	12
2	STATE OF THE ART	15
2.1	Requirements Engineering . . . . .	15
2.2	Model-Driven Development . . . . .	18
2.3	Coexisting Graphical and Textual Representations . . . . .	23
2.4	Model-Driven Development in Requirements Engineering . . . . .	26
2.5	Scope of this Thesis . . . . .	34
<b>II</b>	<b>ANALYSIS OF A SPECIFICATION APPROACH BASED ON COEXISTING ACTIVITY DIAGRAMS AND TEXTUAL REPRESENTATIONS</b>	
3	COEXISTING GRAPHICAL AND TEXTUAL REPRESENTATIONS OF REQUIREMENTS	39
4	A CASE STUDY ON QUALITY ISSUES AND INCONSISTENCIES	57
<b>III</b>	<b>AUTOMATIC GENERATION OF TEXTUAL REPRESENTATIONS FROM ACTIVITY DIAGRAMS</b>	
5	REMOVAL OF REDUNDANT ELEMENTS WITHIN UML ACTIVITY DIAGRAMS	71
6	INFORMATION EXTRACTION FROM HIGH-LEVEL ACTIVITY DIAGRAMS	83
7	CONCLUDING DISCUSSION	93
7.1	Summary . . . . .	93
7.2	Discussion . . . . .	95
8	OUTLOOK	103

#### IV APPENDIX

A	FORMALIZATION OF INCONSISTENCIES AND QUALITY ISSUES	107
A.1	Formal Semantics of the Examined UML2 Activities . .	108
A.2	Formal Definition of the Textual Representation . . . .	109
A.3	Connections between the Activity Diagram and the Textual Representation . . . . .	111
A.4	Categories of Inconsistencies and Other Quality Issues	112
B	ALTERNATIVE TEXTUAL REPRESENTATIONS OF ACTIVITY DIAGRAMS	121
B.1	Group Representation . . . . .	121
B.2	Normal Form Representation . . . . .	122
B.3	Tree Representation . . . . .	124
B.4	Exact Equivalent Representation . . . . .	125
B.5	Evaluation of the Textual Representations . . . . .	126
C	TOOL PROTOTYPE	129
C.1	Structure . . . . .	129
C.2	Overview . . . . .	130
	BIBLIOGRAPHY	131

## LIST OF FIGURES

---

Note that in addition to the figures below, this thesis also contains figures from the original publications.

Figure 1.1	Outline of the main part of this thesis . . . . .	12
Figure 2.1	Excerpt of an RS . . . . .	16
Figure A.1	Document tree of Requirements Specification (RS) in Figure 2.1 . . . . .	109
Figure B.1	Group Representation . . . . .	122
Figure B.2	Normal Form Representation . . . . .	123
Figure B.3	Tree Representation . . . . .	124
Figure B.4	Exact Equivalent Representation . . . . .	125
Figure C.1	Procedure of the text generation from activity diagrams . . . . .	129
Figure C.2	Dialogs of the tool prototype . . . . .	130

## LIST OF TABLES

---

Note that in addition to the tables below, this thesis also contains tables from the original publications.

Table 2.1	Relevant activity elements in the examined methodology . . . . .	22
Table 2.2	Works on model-to-requirements approaches .	29
Table 2.3	Works on requirements-to-model approaches .	32
Table B.1	Ranking of textual representations for each participant in descending order . . . . .	127
Table B.2	Aggregated ranking of textual representations for all participants . . . . .	127

## ACRONYMS

---

BPMN	Business Process Model and Notation
MBE	Model-Based Engineering
MDD	Model-Driven Development
NL	Natural Language
NLP	Natural Language Processing
OEM	Original Equipment Manufacturer
RE	Requirements Engineering
RS	Requirements Specification
SysML	Systems Modeling Language
UML	Unified Modeling Language

## Part I

# REQUIREMENTS SPECIFICATION WITH GRAPHICAL MODELS AND TEXT



## INTRODUCTION

---

In this thesis, we investigate an approach that uses graphical models and textual descriptions side by side for specifying systems and software in an industrial environment. For this purpose, a case of such a specification approach that uses activity diagrams<sup>1</sup> as graphical models and hierarchically structured text as textual representation is examined. The approach is analyzed in detail and challenges identified in the process are addressed. In this chapter, the reasons and motivation for such an approach are presented in Section 1.1. Section 1.2 provides insights into the context of the examined specification approach applied to describe functions of a vehicle system. Section 1.3 states the problems that are addressed in this work. Section 1.4 summarizes the contributions. Section 1.5 provides an overview of the methodological background of this thesis. Finally, Section 1.6 presents the outline of the remainder of this thesis.

### 1.1 MOTIVATION

For embedded and distributed systems the tendency of increasing complexity and further individualization continues to persist [41, 195]. As a result, the development of systems and software requires more and more effort. Requirements Engineering (RE) has been impacted by this tendency because it represents the beginning of the development of a product. This impact on RE is essential for product development, as RE is still considered one of the most crucial phases during development because all subsequent development phases depend on its outputs [105, 237]. A fundamental purpose of RE is the elicitation and management of requirements, typically in the form of requirements specification (RS) documents [65].

These RS documents most commonly consist of textual Natural Language (NL) requirements [164]. Despite the wide adoption of NL to specify requirements, its use has a number of drawbacks (e.g., ambiguity) and limitations that have repeatedly been emphasized [30, 199, 203]. Still, NL text remains the predominant form of documentation for RS documents [126]. In order to mitigate the identified weaknesses of NL text, the use of Natural Language Processing (NLP) techniques has been identified as a possible solution [158]. These techniques attempt to improve the degree of automation of RE tasks in order to

---

<sup>1</sup> Unless indicated otherwise, in this thesis the term *activity diagram* refers to Activity Diagrams of the OMG Unified Modeling Language (UML) Specification Version 2.5 [172].

reduce the required effort and facilitate work that is otherwise often error prone. There are a number of approaches in RE that make use of NLP [163]. Nevertheless, these methods are not perfect and under some circumstance yield worse results than manual approaches [29] (i.e., in safety-critical contexts where each requirement must be manually reviewed anyway). Thus, even considering the recent and ongoing progress in NLP, it can be assumed that NLP does not offer a permanent solution for managing the increasing complexity of RE tasks.

In addition to NLP, it has been suggested that (semi-)structured NL be applied to specify requirements [139, 148]. While this mitigates the problem of ill-structured and badly written requirements, it neither resolves the problem of ambiguity nor eases the handling of the increasing number of requirements. In addition, Weber et al. found that modern systems have reached a degree of complexity such that RE tools that focus on textual requirements alone, are not sufficient for coping with the resulting RS documents [235].

Moreover, Mavin et al. state that NL text used as the only means to express requirements is not sufficient and should be complemented by other notations (e.g., truth tables, mathematical formulas) [150]. Graphical models are another possible notation that could support RE tasks in software and systems engineering [113, 168]. The graphical modeling languages UML [172] and Systems Modeling Language (SysML) [175] have won recognition in practice for use with such RE tasks [187]. Besides their ability to visualize a system's characteristics, graphical models usually provide a well-defined vocabulary of usable elements with (oftentimes) more precisely defined semantics than NL [34]. As a result, models are less affected by the problems inherent to NL, such as ambiguity, and also offer additional potential for automation [86]. Moreover, it has been confirmed that using graphical models for specification improves reusability and analytical capabilities [234].

Despite these advantages, the exclusive use of graphical models is not advisable. As with text, graphical models suffer from a lack of clarity if they contain too much information [14].

Furthermore, it is not possible to go without any text in RE [81]. Understanding NL does not require knowledge of specialized notations. Thus, NL text is recognized as a means that can be easily applied in RE [185]. As a result, NL text enables involving a wide range of stakeholders with various backgrounds and different knowledge, who may otherwise not be capable of understanding the notations of graphical models [14]. Legal considerations are another essential reason for textual representations of requirements [141, 215]. Furthermore, tool support for industrial applications and the exchange of graphical models is not adequately standardized, with the result that



handovers between manufacturers and suppliers are still realized by exchanging textual documents [59, 72].

Ultimately, it can be assumed that graphical models do not replace textual requirements, but rather supplement them [215]. This supplement may take different forms regarding which representation contains what information. Both textual documents and graphical models may represent the same information but display it in a different manner. However, they may also contain different information and complement one another. Finally, a combination of these cases is also possible. In this situation, the representations contain overlapping information and one or multiple representations contain their own unique information.

The argument for the application of coexisting graphical and textual representations is supported by the fact that this approach is perceived as beneficial even outside of RE. In *Business Process Modeling*, Carnaghan reports that the combination of visual and textual representations is superior to the exclusive use of one method [48]. Moreover, Burton-Jones and Meso generally recommend including a textual narrative in addition to visual representations to improve understandability [44].

## 1.2 CONTEXT: REQUIREMENTS ENGINEERING IN THE DEVELOPMENT OF AUTOMOTIVE SYSTEMS

In this thesis, the application of a specification approach that uses coexisting graphical models and textual representations is analyzed in the context of the development of automotive systems. In order to give the reader an understanding of this context, this section provides a short overview of the role of RE in the automotive industry.

In the automotive industry, the trend of innovation being increasingly driven by functions that rely primarily on software persists [97, 103]. Consequently, the importance of software in the development of vehicle systems and components continues to grow as well [186]. As development activities in the automotive industry today take place across various domains (e.g., infotainment and control engineering of mechanical and electronic components), software has become employed in a wide range of applications [204]. Unsurprisingly, different domains require different approaches and methods for development. Because of its significance for software development, this trend also affects RE [109]. As a result, the activities in RE have a substantial impact on the development of automotive systems [40].

At the same time, the automotive industry is characterized by the tendency of subsystems and components being developed by suppliers in order to reduce costs, improve quality, and save time [190]. This distribution of development tasks has led to a situation in which the Original Equipment Manufacturer (OEM) merely specifies the systems

and components that are to be developed. The development itself is then executed by suppliers based on the *RS* that results from the *RE* activities of the *OEM* [183]. The development outcomes of the suppliers then need to be integrated into the vehicle by the *OEM* [39]. Consequently, the *RS* serve as the main means of communication between *OEMs* and different suppliers. Thus the *RS* must ensure that the developed systems and components of distinct suppliers are compatible with one another. For these reasons, the *RS* are important for the success of a development project in the automotive industry [123].

This procedure, combined with aforementioned trends in the automotive industry has stretched the established tools and processes in *RE* to their limits [37]. In order to create high-quality *RS*, the application of Model-Driven Development (*MDD*) approaches has become more popular, and *MDD* is considered as a solution for the future [42, 103]. Nevertheless, for the reasons mentioned in the previous section the use of models alone is not sufficient (e.g., textual *RS* serve as the contractual basis between *OEM* and suppliers). Hence, it appears that the application of graphical models in combination with the established use of textual representations is a suitable method for meeting challenges in the automotive industry [235].

### 1.3 PROBLEM STATEMENT

With the issues associated with the *RS* in mind, the research community has developed a number of approaches that use a combination of graphical models and textual representations [165].

In terms of modeling, these approaches differ regarding the diagram types used or even the modeling languages employed [200, 230], while the text-based representations exhibit differences in the form [86, 199]. Furthermore, there are also differences in the ways in which both representations originate from one another and subsequently, which representation contains what information. It is possible to generate the textual parts of the *RS* from the graphical models [165] as well as to generate the graphical models from the textual parts [7]. In addition, both representations may be manually created independent of one another [14].

As the number of different approaches for combining graphical models and textual representations suggests, there is no standard solution that is suitable for all possible situations. This argument is supported by Bajec et al. [20], who state that in practice it is even necessary to adapt development methods to the specifics of each individual project. In the context of this thesis, this means that it is possible to encounter comparable specification approaches in industry that have never been examined from a research perspective. However, these approaches have been implemented by practitioners with their own problems and organizational- and domain-specific realities

in mind. In order to evaluate the possible applicability of these specification approaches in practice, it is necessary to gain insights into a similar approach that reflects these circumstances, i.e., an approach implemented independently in industry and productively used by practitioners [10]. If these insights are neglected, it cannot be assured that comparable specification approaches will be adopted in practice, let alone used effectively in day-to-day business.

**Problem Statement 1:**

We need an understanding of how a specification approach that uses coexisting graphical and textual representation of requirements is realized and used in practice.

To address this problem, a specification approach is analyzed that employs *UML2 Activity Diagrams* as graphical models and hierarchically structured text as textual representation to describe the behavior of vehicle functions. In the examined case, the textual representation contains all the information that exists in the activity diagram and sometimes additional, more detailed information (e.g., descriptive information of surrounding context). As part of the analysis, we find that the application in practice implies its own difficulties and problems. Concretely, we find that the main difficulties are tied to inconsistencies between representations and to other quality issues with the individual representations.

**Problem Statement 2:**

We need to ensure synchronicity between representations and avoid quality issues in each representation.

This problem statement is addressed by a set of suggestions that, if adhered to, prevent both the occurrence of inconsistencies between the representations and a number of other quality issues. To support the implementation of these suggestions, we develop an approach for the examined use case that automates the generation of the textual representations from *UML2 Activity Diagrams*.

#### 1.4 CONTRIBUTIONS OF THIS THESIS

The contributions of this thesis address the problem statements formulated in the previous section. This section is structured as follows. First, an overview of the contributions of the thesis and how they are connected is presented. This is followed by a short introduction of each contribution. At the end of the section additional contributions are mentioned that do not directly address the problem statements and can thus be seen as byproducts of the developed concepts.

#### 1.4.1 Overview

This thesis features the following contributions:

- I. Analysis of a Specification Approach using Coexisting Graphical and Textual Representations of Requirements in Practice
- II. Categorization and Assessment of Inconsistencies and Other Quality Issues
- III. Suggestions for how to Implement a Specification Approach Using Graphical and Textual Representations
- IV. Removal of Redundant Elements within UML Activity Diagrams
- V. Automatic Generation of Textual Representations from UML Activity Diagrams

Contribution I and II constitute the analytical portion of this thesis. These contributions are dedicated to the analysis of a specification approach that employs coexisting representations in practice. While contribution I aims at eliciting general insights about the application of the specification approach and its implications in practice, contribution II elaborates the problem of inconsistencies between the representations and other quality issues.

The results from the analytical portion of this thesis represent the basis for the constructive portion. Based on the insights obtained in the analytical portion, suggestions are formulated in an attempt to facilitate the application of the approach in a way that realizes its full potential (Contribution III). Among these suggestions is a generative approach for the textual representation that uses the activity diagram and prevents inconsistencies between the representations from occurring. Contribution IV and V address this topic. All of the contributions are discussed in more detail in the following subsections.

#### 1.4.2 *Analysis of a Specification Approach using Coexisting Graphical and Textual Representations of Requirements in Practice (I)*

In the first study of this thesis we systematically analyze the specification approach. This analysis is designed to address **Problem Statement 1** and hence tries to improve the understanding of comparable specification approaches when they are applied in practice. The study builds on a number of interviews conducted with stakeholders of a system of an industry partner. We use the results to identify advantages and disadvantages from the point of view of users. We find that the models are perceived as beneficial and are used in an informal manner. The majority of participants employ the representations for different purposes but see the textual representations as the primary

specification artifact. These insights are integrated into a model that assigns which development tasks are best performed using the distinct representations or a combination of the two. In addition, the information obtained in this analysis serves as the basis for formulating suggestions regarding how to improve the way the specification approach of the industry partner or similar approaches are implemented in practice.

#### 1.4.3 *Categorization and Assessment of Inconsistencies and Other Quality Issues (II)*

The first study also reveals that inconsistencies between the representations and other quality issues related to the representations are perceived negatively by the users of the approach. To gain a better understanding of the impact of these inconsistencies and certain quality issues on development, a further study is presented that analyzes this finding in greater depth. This study also relates to **Problem Statement 1**. For this study, we wanted to obtain knowledge about severity of the inconsistencies and other quality issues that may arise during the application of the examined approach in practice. To achieve this goal, we define nine categories of possible inconsistencies and other quality issues. These categories are partly based on findings from the activity diagrams and their corresponding textual representations, found in the [RS](#) document of the examined system. Further categories are inspired by demands of industry development standards (such as Automotive SPICE [232], CMMI-Dev [51], ISO/IEC/IEEE 29148 [226]). These categories are used to determine how often these deficiencies actually appear in a productively used [RS](#) document. The categories are then assessed by the stakeholders of the system regarding their severity and the possible impact on users' work. A combination of the number of occurrences and the assessment of the categories is used to determine which category of inconsistencies and other quality issues has the most sizable impact on the quality of the specification artifacts and therefore on their use. The results reveal that over 70% of the functions are affected by deficiencies that are seen as major issues by the participants. Thus, we provide empirical evidence that inconsistencies and other quality issues have a serious negative impact on the application of the specification approach in practice. From these results we conclude that a concept is needed to prevent the occurrences of inconsistencies and other quality issues.

#### 1.4.4 *Suggestions for how to Implement a Specification Approach Using Graphical and Textual Representations (III)*

Based on the knowledge obtained through contributions I and II, we develop a set of suggestions that recommend changes in the implementation of the specification approach in order for it to reach its full potential. These suggestions primarily relate to **Problem Statement 2**. The suggestions attempt to ameliorate the problem of inconsistencies and other quality issues that is investigated as a part of contribution II. We argue that the generation of the textual representations based on the activity diagrams is an appropriate measure for preventing these issues. This should be implemented in an automatic manner to avoid mistakes occurring from human action. Moreover, the suggestions also address other weaknesses mentioned by practitioners, including how the artifacts should be designed and at what time and for what tasks the artifacts are suitable.

#### 1.4.5 *Removal of Redundant Elements within UML Activity Diagrams (IV)*

In order to enable automatic generation of textual representations as mandated by contribution III, it is necessary that the activity diagrams contain only unique elements (in particular *ExecutableNodes*). This is required, as the redundant elements would impede the unambiguous identification of execution paths and of propositional logic relations of the elements in the activity diagram. This contribution represents a prerequisite for addressing **Problem Statement 2**. To ensure this prerequisite is met, a transformation is developed that removes multiple *ExecutableNodes* from UML activities. This is achieved by merging redundant elements into a single element and adding additional *ControlNodes* and edges.

#### 1.4.6 *Automatic Generation of Textual Representations from UML Activity Diagrams (V)*

By using the redundancy-free activity diagrams as inputs from contribution IV, it is possible to automate the generation of the textual representations from the activity diagrams as suggested by contribution III. The activity diagrams are employed as the basis for extracting the information for the textual representation in an automatic manner. The extracted information can then be used to create different textual representations for the RS document. This procedure guarantees that both representations are consistent and prevents a number of other quality issues.

#### 1.4.7 Contributions beyond the Problem Statements (VI)

The contributions IV and V offer concepts that are applicable beyond the problem statements formulated in Section 1.3. Contribution IV presents an algorithm that creates reachability graphs for the examined activity diagrams. While this algorithm does not address any of the problem statements, it is needed to demonstrate that the behavior of the UML activity remains unchanged by the redundancy removal for the assumed semantics. The comparison of these reachability graphs (which in itself also represents another contribution) is used to demonstrate that the behavior of the activities is in fact preserved.

Contribution V relates to the extraction of information from the activity diagrams in order to generate textual representations automatically. This information can also be used to create test cases or to support other development tasks.

### 1.5 METHODOLOGICAL BACKGROUND

In information systems research there are two prevalent paradigms, behavioral science and design science [108]. This thesis utilizes methods from both paradigms but places a strong focus on design science, as this is mainly seen as a problem-solving paradigm and this is also the main theme of this thesis.

Hevner et al. propose seven guidelines on how to conduct, evaluate, and present design science contributions [108]. These guidelines provide a framework that helps researchers carry out their efforts in an effective and efficient manner. The following list presents these seven guidelines and describes how they are met by the contributions of this thesis.

1. **Design as an Artifact.** A set of suggestions and a tool prototype are the created artifacts of this thesis. The suggestions offer rules for how to realize a specification approach using coexisting graphical and textual representations in practice. Furthermore, the application of the suggestions in combination with the tool prototype, prevents problems of inconsistent representations that might occur in practice.
2. **Problem Relevance.** The contributions address problems voiced by practitioners and fill gaps identified by the research community. The relevance of the main problems addressed in this thesis is assessed in a systematic way.
3. **Design Evaluation.** The functioning and applicability of the solutions are demonstrated using the specification artifacts of a real-world system specification.

4. **Research Contributions.** The contributions of this thesis are introduced in Section 1.4. They include an analysis of an industrial realization of a specification approach as well as a method for generating textual representations from activity diagrams.
5. **Research Rigor.** This thesis is based on previous work in the areas of RE and MDD (see also Chapter 2). Existing knowledge is continuously referred to throughout this thesis.
6. **Design as a Search Process.** This thesis begins with the exploration of implications of an industrial application of a certain type of specification approach. The issues identified are subsequently elaborated on and addressed.
7. **Communication of Research.** The results of this thesis have lead to a number of peer-reviewed publications that are publicly available (see p. vii).

Meeting these seven guidelines demonstrates that this thesis adheres to research standards and contributes to the field of information systems.

## 1.6 OUTLINE

This thesis is structured in the following manner. This chapter (Chapter 1) is followed by Chapter 2 *State of the Art*. Chapter 2 presents the foundations of topics relevant to this thesis and explanations of the terminology used (Section 2.1 and 2.2). It also features an overview of work related to the topic of coexisting graphical and textual representations.

	Chapter Name/Publication	Problem Statement	Contribution
Chp. 3	Coexisting Graphical and Structured Textual Representations of Requirements: Insights and Suggestions	1 & 2	I & III
Chp. 4	A Case Study on a Specification Approach using Activity Diagrams in Requirements Documents	1 & 2	II & III
Chp. 5	Removal of Redundant Elements within UML Activity Diagrams	2	IV & VI
Chp. 6	Information Extraction from High-Level Activity Diagrams to Support Development Tasks	2	V & VI
	Analytical Contribution	Constructive Contribution	

Figure 1.1: Outline of the main part of this thesis



Chapters 3, 4, 5, and 6 constitute the main part of this thesis. Figure 1.1 illustrates the order of the chapters and their respective titles. In the third column in the figure describes which problem statements are addressed in each chapter. The last column clarifies which contribution stems from which chapter.

The main part of this thesis consists of four chapters. Each chapter represents one publication by the author of this thesis. Chapters 3 and 4 contain the analytical contributions as well as some constructive contributions. Such contributions are also present in Chapters 5 and 6, which solely contain constructive contributions. The beginning of each chapter provides information about the publications followed by an explanation of how the publication relates to the context of this thesis, followed by the publication itself.

After the main part, the contributions of this thesis are summarized and discussed in their entirety in a *Concluding Discussion* presented in Chapter 7. The last chapter, Chapter 8, provides an *Outlook* regarding possible future work.



## STATE OF THE ART

---

This thesis examines the practical application of graphical models and textual representations for specifying requirements. Consequently, this thesis touches aspects of requirements engineering (RE) and model-driven development (MDD). This chapter provides an overview of these topics and explains the terms and concepts that are needed to understand the contributions of this work. In addition, related work on the use of coexisting graphical and textual representations is summarized. Using this summary, open points are emphasized that will be addressed in this thesis.

First, Section 2.1 provides an overview of the concepts in RE that are relevant for this thesis. Section 2.2 offers insights into the topic of MDD and describes how this thesis relates to the topic. Section 2.3 presents an overview of work that uses coexisting graphical and textual notations, and Section 2.4 places these coexisting graphical and textual notations into the context of RE and MDD. Finally, Section 2.5 explains how this thesis relates to these topics and how the contributions address gaps in research.

### 2.1 REQUIREMENTS ENGINEERING

Requirements Engineering (RE) is defined by the International Requirements Engineering Board as [32]:

A systematic and disciplined approach to the specification and management of requirements with the following goals:

1. Knowing the relevant requirements, achieving a consensus among the stakeholders about these requirements, documenting them according to given standards, and managing them systematically,
2. Understanding and documenting the stakeholders' desires and needs,
3. Specifying and managing requirements to minimize the risk of delivering a system that does not meet the stakeholders' desires and needs.

The importance of RE stems from the fact that it is the basis for all downstream development phases [79]. Hence, RE is considered to have an important impact on the outcome of development projects and at the same time is one of the biggest challenges [15].

Consequently, the quality of requirements is of high importance for RE. A requirement is a “*statement which translates or expresses a need and its associated constraints and conditions*” [226]. A common method for capturing these statements is in the form of written NL text [126]. While NL presents an easy way to capture requirements that requires little initial effort [185], it is not without difficulties [30] and is therefore often considered not satisfying for RE [215].

To address these difficulties, an effort has been made by the research community to ensure a uniform structure of the requirements in order to improve their overall quality. For this purpose syntax patterns have been proposed [64]. A well-known representative of a syntax pattern is EARS [148]. When both the usable syntax (sentence structure) and also the vocabulary (words) that can be used are restricted, the result is a controlled natural language [240]. If there is no longer a recognizable sentence structure, however, the controlled language is not even considered to be a NL. The case studies in this thesis feature a textual representation that uses a restricted vocabulary in combination with a hierarchical structure to arrange the text. This type of representation is referred to as structured text.

Despite the efforts towards adopting artificial languages, requirements are still predominantly supplied as documents containing NL text [72, 215]. These documents are known as (software/system) Requirements Specifications (RS) and represent a “*structured collection of the requirements (functions, performance, design constraints, and attributes) of the software/system and its operational environments and external interfaces*” [226]. An excerpt of an RS is depicted in Figure 2.1. The figure displays characteristics that are relevant in the context of this thesis.

ID	Text	Level	Type
1234	<b>Chapter</b>	1	Heading
1235	Section	2	Heading
1236	Requirement 1	3	Requirement
1237	Requirement 1.1	4	Requirement
1238	Requirement 1.2	4	Requirement
1356	Requirement 2	3	Requirement
1357	Information Requirement 2	4	Information

Figure 2.1: Excerpt of an RS

An RS consists of distinct entries that have a number of attributes. As such, an RS takes on a table-like appearance where the rows represent an entry and the columns represent the attributes of an entry. Each attribute has a specific data type (e.g., string, enumeration, numerical value). These attributes are, among other functions, used to

provide other mandatory characteristics necessary for RE tasks. The *ID* attribute in Figure 2.1 ensures that each entry is uniquely identifiable. The *Text* attribute contains the text of the requirements or additional text to either structure the document (*ID 1234* & *ID 1235*) or to provide further information (*ID 1357*). The *Level* attribute is needed to create a hierarchical document structure. Because of their importance these attributes are called *inherent* attributes, as they must always exist. In addition, there are further mandatory characteristics of RS (e.g., the possibility to create baselines). However, these are not relevant for this thesis and are therefore not depicted in Figure 2.1. All the aforementioned properties are required by standards such as the ISO/IEC/IEEE 29148:2011 [226] (see also [233]). Additionally, Figure 2.1 displays the attribute *Type*. This attribute illustrates that there may be other attributes that can be freely defined to meet specific needs. Here, the *Type* attribute is used to define what kind of entry is in the RS.

Due to these highly specialized characteristics of RS and the single requirements therein, there is a need for tools to support RE related tasks [214]. This has led to the development of a plethora of available products (see [59] for an extensive survey on RE tools). Consequently, while these characteristics must be implemented by RE tools [119], they might be realized in different ways. For instance, some tools force all entries in an RS to have the same attributes (e.g., IBM DOORS [115]), while others allow entries with different attributes to exist within an RS (e.g., ProR [70]). The case studies in this thesis encompass RS managed by the tool IBM DOORS. However, the conclusions of this thesis are not restricted to the data model of IBM DOORS and can be applied so long as it is possible to recreate the properties mentioned (which is the case for all established RE tools).

Another relevant concept related to RE in this thesis is *traceability*. Traceability is defined by Gotel and Finkelstein as “*the ability to describe and follow the life of a requirement, in both a forwards and backwards direction, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases*” [96]. Traceability has been considered an important factor in the design of complex software systems for quite some time [194]. Nevertheless, it is still insufficiently implemented [13, 102] because it requires significant effort for even moderately sized systems [193]. Hence, it is sometime debatable whether the effort invested into traceability yields a benefit [13].

Despite benefit considerations, there is an obligation in safety-critical development projects to provide evidence regarding safety issues in the form of traces [95]. In some domains, traceability is mandated by regulations (such as in the automotive industry [220], aeronautics [6], and in medical devices [152]) and software process improvement standards (e.g., Automotive SPICE [232], CMMI-Dev [51]). Un-

surprisingly, each domain has its own unique challenges regarding traceability [146].

Although research on traceability has been largely driven by the RE community [50], it is also a relevant topic in other research areas. Model-Driven Development (MDD), for example, has been identified as a further area of application for traceability [49]. Because of the tendency to incorporate models for RE purposes, traceability is an issue that not only affects RE and MDD, but can also help bridge the gaps between them [238].

## 2.2 MODEL-DRIVEN DEVELOPMENT

Model-driven development (MDD) denotes *"the systematic use of models as primary artifacts during a software engineering process"* [113]<sup>1</sup> where *"a model is an abstraction of some aspect of a systems"* [88]. Although these definitions relate to the application of models in software engineering, models have been in use in other domains for quite some time. In physics and chemistry different types of particle models are used to describe matter, while in electronics engineering circuit diagrams are commonly used. Multiple domains visualize their systems through technical/engineering drawings. All of these models are used to make predictions about circumstances in the real world and serve different purposes. Hence, they reflect real-world properties at different levels of detail.

While models are often rendered in graphical form (such as the aforementioned engineering diagrams), these graphical forms are merely the representation that is easier to grasp [17]. The underlying concepts and their relationships can always be captured in textual form. Moreover, in cases where the models are created and managed with the help of computers, textual forms must exist to enable the processing [188] – even for models that are primarily represented in graphical form. In addition, there are models that do not have graphical representations at all, such as controlled languages [240]. Also, the underlying structure of RE tools represents a data model [59] (see Section 2.1). Therefore, modeling languages (e.g., UML) often distinguish between the models themselves and their representation, which may be graphical or textual.

In MDD, models are employed in an engineering process in order to develop a product. At the same time, this engineering process involves numerous engineering domains. These domains involve various tasks during different development steps. Consequently, there is a wide range of modeling languages and model types that have dif-

<sup>1</sup> Closely related research areas such as Model-Based Engineering (MBE) rely on models to a lesser extent than MDD. In MDD models are the primary artifacts while in MBE they may simply support engineering tasks [225].

ferent aims and possible areas of application. As a result the created engineering models can be classified according to different criteria.

Eigner et al. suggest a classification that distinguishes three “levels” of models that differ in their purposes and level of abstraction [72]. Their first level, denoted as *specification models*, includes models that are solely descriptive and are used to specify behavior or logical structure. The second level features models used for simulation and validation that involve multiple disciplines. The third level, *discipline-specific models*, addresses models that, as their name suggests, focus on modeling aspects of certain disciplines (e.g., the aforementioned engineering drawings such as circuit diagrams in electronics engineering [118] or CAD models [210] and thereof derived drawings for geometric dimensioning and tolerancing of mechanical parts [120]). Because of the many domains, there is a plethora of discipline-specific models that support a wide range of tasks. For instance, in mechanical engineering they can act as the basis for deriving bills of materials or in electronics engineering they can be used to layout circuit boards.

While the classification of Eigner et al. considers multiple engineering disciplines, this thesis focuses on the software engineering process. A classification of software models created by France and Rumpe [88] classifies models into *development models* and *runtime models*. Here, the distinction lies in the level of abstraction of the model. While *development models* are “above code level”, *runtime models* represent “aspects of an executing system”. A further classification made by Störrle follows Fowler’s distinction for different types of UML models [87] and distinguishes software models with respect towards their degree of formality into *informal*, *semi-formal*, and *fully formal* models [222]. *Informal models* “support communication and cognition”. *Semi-formal models* “support design and documentation activities” and *formal models*, among other functions, “allow simulation and generation of code and test cases”. In practice it has been shown repeatedly that the majority of users employ *informal* and *semi-formal* models [68, 222].

Although these classifications use different aspects for distinction, they do exhibit commonalities. In particular, the idea of Störrle’s *formal models* used for simulation and generating source code can be found in the other two classifications (Eigner et al. – *second level*, France and Rumpe – *runtime models*). It can be seen that the tasks a model is used for are connected to the model’s formality to a certain degree. For this reason, modeling languages support a varying degree of formality. As a result, it is not possible to undisputedly map modeling languages to single classes of models.

Nevertheless, Eigner et al. map modeling languages to the levels within their own classification. Their *specification models* consist of models on behavior and logical structure. The modeling languages mentioned for these types of models are UML and SysML. Aside from engineering models, modeling languages for business processes (e.g.,

Business Process Model and Notation (BPMN) [171]) may also fall into this category of models as they also specify behavior and logical structure. For the models on their second level, Eigner et al. mention Matlab/Simulink [227] or Modelica [160]. This is logical, as Matlab/Simulink's and Modelica's primary purpose is the simulation of potentially interdisciplinary models. Still, depending on the degree of formality, simulation is also achievable with UML and SysML models (see also Foundational UML [169]). At the same time, it cannot be denied that models created with Matlab/Simulink or Modelica also constitute models on behavior and logical structure. As a result, it is possible to conclude that the lines separating classes of models are inherently blurred, since classification is dependent on the intentions with which the model is created. Accordingly, the tasks performed with software engineering models also cannot be clearly mapped to certain degrees of formality.

Among other functions, software engineering models can be employed for pure visualization purposes [107], to conduct Failure Mode and Effects Analysis (FMEA) [217], to generate source code [231], and to create test cases [38]. Pure visualization is possible with all models that have graphical representations regardless of their degree of formality. The creation of test cases requires at least semi-formal models although the degree of possible automation increases with the degree of formality. Since source code is a formal language, it is not surprising that its automated generation relies on fully-fledged formal models.

UML supports all of these specificities of software models and hence has become widespread in software engineering [68]. It is often even considered the de facto industry standard for software modeling [167]. At the same time, there are various adaptations and subsets of UML that are designed with different kinds of systems and levels of detail in mind [106]. The methodology examined in the following studies of this thesis employs a subset of UML2 activities with a predefined structure to describe the activation of functions of a system.

### UML2 Activities

According to the *OMG Unified Modeling Language Specification, Version 2.5*, activities are one of the model types used to describe behavior [172, p. 283]. As such, the specification defines activities as control and data flow models. For RE, the use of activities (and their graphical representation activity diagrams) has been assumed to be helpful for a long time [14]. Hence, it comes as no surprise that by now they are intensively used for the specification of software requirements [99, 196]. In addition to facilitating software engineering, activities offer concepts and constructs that enable modeling for a wide variety of domains [33]. This makes activities also suitable for application in



Business Process Modeling [202] as well as for specifying hardware design [209], although UML and, therefore activities were originally intended to model software systems.



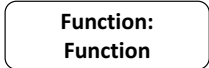
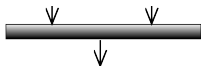



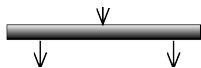

At the same time, the versatility of activities is not without consequences. Bock notes in the companion series [33] to the UML2 Specification that *"UML2 activity and action models are defined independently of application, some features are more appropriate to some domain styles than others"*. Consequently, *"redundancy is unavoidable when creating an abstraction over user groups that do not overlap"*. As a result, most approaches in practice limit the set of used model elements and introduce other restrictions or rules for their UML activities depending on the application [208]. In this thesis, a methodology of an industry partner is examined that uses UML2 activities to specify the activation of functions of an automotive system. These functions are modeled by using a subset of the available elements of activities and by imposing a predefined pattern. This pattern resembles a pattern used to model textual requirements [141]. In particular, the activities describe the activation of functions by a combination of triggers, conditions, and their connections (for more detailed information, please see Chapter 3 and Chapter 4). The elements relevant for this thesis are shown in Table 2.1 – all of these are among the most commonly used constructs of activities [197]. Only the first seven elements are part of the UML activities of the studied methodology. The last two elements (below the horizontal line) are needed for the concept presented in Chapter 5.

The first column shows the graphical symbols of the elements. Note that *JoinNodes*/*MergeNodes* are depicted using the same symbols as *ForkNodes*/*DecisionNodes*. Both (*JoinNodes* and *ForkNodes* as well as *MergeNodes* and *DecisionNodes*) are distinguished by the number of incoming and outgoing edges. *JoinNodes* and *MergeNodes* have multiple incoming edges and one outgoing edge while the opposite is the case for *ForkNodes* and *DecisionNodes*.<sup>2</sup> The second column provides the class names of the elements according to the UML2 specification. These class names can be used as further reference for detailed descriptions of the elements. The third column describes how these elements are employed in the context of the examined methodology.

The first three elements are used in a specific way. The *AcceptEvent-Action* is first labeled with the prefix *Trigger:* followed by the string that specifies the event that executes the element. Only two kinds of *OpaqueActions* exist in the examined activities. The first is used to check whether certain conditions are fulfilled. These checks are

<sup>2</sup> Single incoming and outgoing edges lead to a construct that is equivalent to a single edge. In case of multiple incoming and outgoing edges, implicit *ForkNodes* and *JoinNodes* exist. The use of implicit *ForkNodes* and *JoinNodes* in combination with *MergeNodes* and *DecisionNodes* may lead to a situation in which it is no longer possible to distinguish between a *MergeNode* with an implicit *ForkNode* and a *DecisionNode* with an implicit *JoinNode* on a purely visual level. See the UML2 specification for more details on implicit *ForkNodes* and *JoinNodes* [172, p. 373, p. 387].

Table 2.1: Relevant activity elements in the examined methodology

Symbol	UML Class	Use
	Accept Event Action	Starts the execution of an activity if the specified event occurs
	Opaque Action	Checks whether the specified condition is fulfilled
	Opaque Action	Represents the function to be activated
	Join Node	Propositional logic AND connection of incoming flow
	Merge Node	Propositional logic OR connection of incoming flow
	Flow Final	Ends the incoming flow
	Activity Final	Ends the execution of the whole activity
	Fork Node	Splits a flow into multiple concurrent flows
	Decision Node	Chooses between outgoing flows

denoted by a label starting with the prefix *Check:* followed by the condition checked. The second use of *OpaqueActions* is the function to be activated. Its label consists of the prefix *Function:* followed by the name of the function. These three elements are used only in this specific way in the context of the methodology examined in this thesis. Thus, one can say that this use of elements and the pattern constitute a domain-specific language designed to describe the activation of a function. Aside from these notational peculiarities, the behavior of the elements is the same as intended by the [UML2](#) specification. The same applies for the remaining elements. The *ControlNodes* are used as defined by the [UML2](#) specification and are not subject to restrictions regarding the notation.

Despite their suitability for many applications, activities exhibit a number of problems [208]. One important issue is that the UML2 specification does not provide a formal definition of its semantics [221] and, merely describes activities semi-formally as “Petri net-like graphs”. Thus the data flow in activities may be intuitively interpreted as it is in Petri nets [180]. However, there are cases that require modifications or do not allow mapping from UML2 activities to Petri nets in any way [223]. The semi-formal definition allows for these “semantic variations”, which can be considered an advantage since it enables alternative behavior during runtime without explicitly recording these choices in the model itself [33]. These “semantic variations” cause “a model in one implementation to execute differently than the same model in another implementation, with no standard way to tell what the differences are” [33, p. 46]. At the same time, this results in ambiguity problems that aggravate automation in the development process [58]. In response to this, there have been many approaches that formally define semantics for UML2 activities (i.e. [98, 219]) – not all of which use Petri nets as their semantic domain. Nevertheless, there is still no approach that addresses all aspects of activities [90]. Thus, it is considered useful to reduce activities into simpler constructs depending on their application [132]. The methodology examined in this thesis makes use of the “requirements-level semantics” of Eshuis and Wieringa [76].<sup>3</sup> In these semantics, “The system reacts immediately and infinitely fast to the reception of events. Since the system is infinitely fast, no other event can occur while a system is reacting to an event” [76, p. 440]. Hence, these semantics dismiss aspects that may occur if the execution of actions takes time. As a result, the semantics focus on propositional logic and execution sequences while neglecting aspects of runtime behavior such as concurrency and asynchronous events. In terms of the classification of models, the examined activities can be placed among *informal* or *semi-formal* models as they are above code level. More details on the semantics of activities are provided in the relevant Chapters 5 and 6.

## 2.3 COEXISTING GRAPHICAL AND TEXTUAL REPRESENTATIONS

Although modeling languages might offer a more precise way to describe systems, NL is more accessible since virtually everyone is able to understand it. For this reason, it is commonly used despite its potential drawbacks (see Section 2.1). Moreover, models abstract from real-world circumstances and are often designed for certain purposes, which means that they are less versatile than NL. With these consider-

<sup>3</sup> It must be noted that different variations of these semantics are used for the contributions in this thesis. This is due to the fact that the contributions have different scopes (e.g., applicability (Chapter 5) vs. ease of use (Chapter 6)). As a result, there is no uniform formal definition of activities that applies across the entire thesis.

ations in mind, the idea arose to use more formal notations (such as models) and NL text side by side in a coexisting manner. Furthermore, one is not limited to pure NL text. As shown regarding RE in Section 2.1, there are applications that benefit from restricted vocabulary and predefined sentence patterns, such as those seen in controlled languages, which may still resemble NL text, or even structured text which does not resemble NL.

The first appearance of systematically using text in combination with a more formal representation is Knuth's proposal of *Literate Programming* [127]. *Literate Programming* attempts to facilitate the understandability of source code by "explaining to human beings what we want a computer to do". As part of *Literate Programming*, the author of a source code is supposed to describe comprehensibly what the program does. The proposal uses NL and moreover gives the author of the source code a high degree of freedom. As a consequence, it is not possible to determine exactly in what way the complementary text and the source code relate to one another. The descriptive text may solely paraphrase the source code and therefore contribute no new knowledge to the existing information. However, it may also be possible that additional (e.g., contextual) information is provided in the descriptive text. Also, because of the high degree of freedom the descriptive text may omit information included in the source code.

Overall, a variety of possibilities are imaginable for how different representations might relate to one another.

1. The representations overlap in the content they describe, but one or multiple representations contain content that is unique to one single representation.
2. The representations express the same content.
3. The representations express content that does not exist in the other representations.

The choice of how the representations are combined as well as the types of representations selected depends on the actual application. This thesis focuses on coexisting graphical and textual representations as the studied methodology uses coexisting UML2 activity diagrams and structured text. Thus, in the following sections, related work on coexisting graphical and textual representations is discussed with an emphasis on models that are primarily used in a graphical form in combination with some form of text.

One of the first instances of using text as an addition to graphical models is *Literate Modelling*, presented by Arlow et al. [14] and inspired by Knuth's proposal of *Literate Programming* [127]. *Literate Modelling* can be understood as the application of Knuth's idea to MDD, using models instead of programming code as the main software development artifact. Both *Literate Modelling* and *Literate Pro-*

*gramming* claim to improve the comprehensiveness of the main artifact by adding explanatory text, but unlike *Literate Programming*, the main intention of *Literate Modelling* is to provide additional detailed contextual information. This is meant to enhance business models and in the examined case of Arlow et al. aims to facilitate requirements review. Thus, the focus lies on employing a textual description to complement the models. *Literate Modelling* as intended by Arlow et al. can be attributed to approaches where the content overlaps, but in which the text and the models also contain their own unique content (see enumeration item 1). Another representative of the approaches adding explanatory text is Eriksson's proposal, which aims to combine text with ontologies [75]. Eriksson's intentions are the same – to provide additional detailed contextual information.

Approaches that use multiple representations that express the same content try primarily to improve levels of understanding (enumeration item 2). Using various forms of information representation at the same time in order to improve levels of understanding is known in cognitive theory as multimedia learning [151]. Gemino investigates the use of models in combination with narrations and animations in this context [91]. He finds no considerable advantage for either the animations or textual narrations. Nonetheless, he concludes that supplementing models with one of them may be advantageous to support the use of models. Burton-Jones and Meso confirm this conclusion and recommend including complementary forms of information (such as a textual narrative) in addition to models [44]. In contrast to these findings, at least with regard to the understandability of process models, Rodrigues et al. have found evidence that by themselves, graphical models only marginally improve understandability in comparison to text presented alone [201].

There are a number of works that make use of exact representations of the same content. For instance, Object-role modeling is a method for conceptual data modeling that is represented in diagrams, but also offers an equivalent representation of NL text [104]. Closer to the topic of this thesis, Bolloju and Sun [35] propose using activity diagrams to visualize textual use case narratives. Fockel and Holtmann [86] suggest a methodology using various SysML diagram types and controlled NL. Representations derived in an automatic manner from another representation also fall into the category of representations expressing the same content. For instance, Håkan and Heldal propose a technique to automatically generate textual descriptions for UML class diagrams [43] and Rabiser et al. use variability models to derive product documents in a software product line context [189]. Generation of models from text is also addressed by numerous studies. Epure et al. propose generating process models from NL text [74], Meziane and Vadera present a method for obtaining entity-relationship diagrams from NL [156], and Raj et al. derive

UML diagrams from structured NL [192]. Such generative techniques in the context of RE are presented in more detail in the next sections. Although an automatically generated representation can at best only contain as much information as the representation it is derived from, these techniques may be embedded in more sophisticated approaches. These approaches intent to add further information to the derived representation manually after it is created, which would lead to a form of coexistence as described by enumeration item 1.

Situations in which representations only express content that does not exist in the other corresponding representations is more theoretical in nature. At the very least there are references that connect the representations to one another to a certain degree, which then leads to circumstances already reflected by enumeration item 1. Moreover, one might even argue that the term "representation" does not encompass the display of disjunctive content.

Regardless of how different representations relate to one another, graphical and textual representations may fulfill different purposes (see [198] for a study and an overview of this issue). Graphical models are often described as a more appropriate form of representation for managing complexity [8, 67], but text is a form of representation that can be understood by almost everyone – a reason for its use that is also mentioned in the RE context [19]. Still, storing even equivalent information in multiple forms presents a number of drawbacks and may lead to serious problems such as inconsistencies [1].

Van der Aa et al. dedicated a number of studies to this issue (i.e., [2, 3]). They analyze this problem in the domain of process models and provide a method for detecting inconsistencies between coexisting process models and their textual descriptions. Also relating to process models and textual descriptions, Sánchez-Ferreres et al. use NLP techniques to extract textual elements contained in both representations and align model and textual description by using an optimization [205].

## 2.4 MODEL-DRIVEN DEVELOPMENT IN REQUIREMENTS ENGINEERING

Because of the potential for employing models [41], RE is shifting more and more from text-based specifications to model-based specifications [77, 138]. This model-based representation of requirements is also known as *Requirements Viewpoint* by the SPES Modeling Framework [56]. This concept also appears in other related work on MDD [72, 111]. Despite the many advantages of expressing requirements in this model-based (and accordingly, often graphical) manner, the software engineering industry faces many challenges in the adoption of MDD practices in general [162] and thus also for RE [161].



For instance, representatives of the MDD community (e.g., Eigner et al. [72], France and Rumpe [88], Hutchinson et al. [113]) promote the idea of relying on models as the primary artifacts of a software engineering process. However, in industry it is often not possible to start development projects anew with a clean slate. In fact, the issue has already been recognized at the beginning of MDD and a step-by-step integration with legacy environments and systems has been advised [211]. Many years later, the integration with legacy environments is still considered a success factor for adopting MDD practices in industry [114, 239]. Aside from considering legacy artifacts, text is considered an essential form to represent requirements (see also Chapter 1). Based on these considerations, the combination of text-based requirements and graphical models is seen as a feasible solution for RE [215].

A number of approaches have been proposed that attempt to facilitate the combination of textual and graphical representations of requirements. In a literature review Nicolás and Toval proposed a distinction for approaches that generate textual requirements from models that consists of two groups [165]:

- Generative approaches
- Integrative approaches

Generative approaches include algorithms, rules, or patterns used to generate textual requirements from models. Integrative approaches encompass suggestions and guidelines on how to relate models and textual requirements to one another and are used for the capture of requirements or the generation of RS documents without specified rules. Nevertheless, both types of approaches aim to capture requirements by utilizing models. Thus, the distinction between generative and integrative approaches is not entirely clear. Generative approaches might have an underlying idea of how models and textual requirements relate to one another aside from mere generation, while integrative approaches may employ generative approaches to avoid inconsistencies or to benefit from automation.

Moreover, in cases of coexisting textual and graphical representations of requirements, the generation of models from textual requirements cannot be neglected. Aside from RE in a more general context, Czarnecki and Helsen [54] make a distinction between text-to-model transformations and model-to-text transformations. Model-to-text transformations use the model as the origin for generating the text, while text-to-model transformations use the text as the origin. In the categorization of Czarnecki and Helsen, text is perceived as a concatenation of strings. In this thesis, however, text may exist in more sophisticated forms (e.g., as structured text – see Section 2.1). These sophisticated forms can be regarded as a model of their own. This in turn means that all these approaches are ultimately model

transformations [154]. Nevertheless, for better differentiation, text is considered as such even if there is an underlying model. This means that text encompasses NL as well as formal textual notations with or without a hierarchical structure and all combinations thereof. However, although every kind of model has a textual data structure, these are not considered to be text. In cases where a visual representation is primarily used by stakeholders, this representation is considered to be a graphical model. In simple terms, following common sense, in this thesis everything that is mainly perceived as a series of text in a wider context is regarded as text, while everything represented primarily in a graphic form with symbols (such as arrows and boxes) is considered to be a model. Although substantial parts of this thesis focus on the visual aspects of models, the following review on related work in next subsections also takes into account transformations that generate textual requirements or a structure for textual requirements from non-visual models (or vice versa).

As the related works in this field do not only encompass pure model transformations, but also ideas for how to use the artifacts, the works are henceforth referred to as approaches. In the following sections, approaches that somehow aim to capture textual requirements by employing models are presented together and approaches that address the generation of models from textual requirements are presented separately. First, studies on model-to-requirements approaches are presented followed by requirements-to-model approaches.

#### 2.4.1 *Model-to-Requirements Approaches*

Model-to-requirements approaches use models as a basis to systematically capture requirements. Their application is seen as a method for reducing the effort required when writing requirements and for improving completeness [52, 165]. They also aim to improve the understandability of models by combining graphical models and textual requirements, which in turn aims to facilitate RE tasks [133].

An extensive, systematic literature review on comparable approaches was conducted by Nicolás and Toval. This review dates back to 2009 [165]. The authors provide a comprehensive review of the works addressing model-to-requirements approaches at the time. In their conclusion, Nicolás and Toval argue that the supplement of models with text for RE purposes is an excellent idea. However, they also observe that the topic appears to lack relevance in practice and hence is often not validated in practice.

A more current collection of the research on these approaches is presented in Table 2.2. The first column lists five different groups of origin models. The second column lists the author(s) of the works. The third column provides information on how the approach has been validated (no validation/academic/industrial/in practice). Note



Table 2.2: Works on model-to-requirements approaches  
(NR (Not Reported), ACS (Academic Case Study), ICS (Industrial Case Study), IP (Industrial Practice))

Origin Model	Author	Validation	Gen.
Business Models	Türetken et al. [230]	ICS	✓
	Cox et al. [53]	ICS	—
	González and Díaz [60]	NR	✓
	Cardoso et al. [47]	ICS	✓
	Coşkunçay et al. [52]	ICS	✓
	Li et al. [137]	ICS	—
	Malik and Bajwa [144]	ACS	✓
	Leopold et al. [133]	ICS	✓
	Aysolmaz et al. [18]	NR	—
	Aysolmaz et al. [19]	ICS	✓
Goal Models	Yu et al. [241]	ACS	—
	Antón and Potts [12]	ICS	—
	Jungmayr and Stumpe [124]	ACS	✓
	Letier and van Lamsweerde [135]	ACS	✓
	De Landtsheer et al. [61]	ACS	✓
	van Lamsweerde [130]	IP	✓
	Maiden et al. [141]	ICS	✓
	Alrajeh et al. [9]	ACS	✓
UML/SysML Models			
UML Diagrams	Arlow et al. [14]	ICS	—
Use Case Diagram	Berenbach [26, 27]	ICS	✓
Class Diagram	Meziane et al. [155]	ACS	✓
UML1 Activity/Message Sequence Charts	Drusinsky [69]	NR	✓
Statecharts/Block Diagram	Robinson-Mallett [200]	ICS/IP	✓
UML2 Activity	<b>This work</b>	ICS	✓
Stories, Scenarios, and Use Cases	Maiden et al. [142]	ICS	✓
	van Lamsweerde and Willemet [131]	ACS	✓
	Mavin and Maiden [147]	ICS	✓
	Daniels and Bahill [55]	ACS	—
	Firesmith [83]	ACS	✓
	Maiden and Robertson [143]	ICS	✓
	Cabral and Sampaio [45]	ICS	✓
Unspecified	Firesmith [82]	NR	✓

that in some cases, a combination of validations has been performed. In those cases only the validation that is closest to practice is mentioned (e.g., between academic and industrial, only industrial is listed). The fourth column states whether the approach encompasses generative techniques to derive requirements in a textual form. A ✓ indicates that such capabilities exist while a — indicates that they do not. This refers to whether algorithms, rules, or patterns are provided to derive textual requirements. The presented works all consider human beings to be the users of the results. As a consequence, generative transformations such as MOFM2T [170] (a template-based approach for *Meta Object Facility* (MOF) models), that intend to derive source code or other highly formalized texts are out of scope of this thesis.

From the multitude of works on different types of models, it can be seen that there is no approach that encompasses all features or diagram types of a modeling language. Moreover, approaches that offer generative techniques to derive textual requirements often only use a subset of the capabilities available. As this work examines a specification approach that uses a UML diagram type, the group for UML diagram types is broken down to the level of diagram types. Aside from the proposal, *Literate Modelling*, of Arlow et al. [14], all other approaches focus on one, or at a maximum two diagram types. *Literate Modelling* aims to complement UML diagrams with explanatory text in general, and as such is a concept that can be applied to all UML diagrams. At the same time, addressing all UML diagrams is only possible, since it does not suggest rules for how to construct the explanatory text, which makes it independent of diagram types. As a consequence, *Literate Modelling* is not a generative technique and as such does not offer the possibility of automation. The focus of generative techniques on certain capabilities of a modeling language is logical, as this reflects the realities in practice [132] and represents a research challenge on its own [153]. Because of project- and domain-specific needs, it does not appear to be worth the effort (or perhaps even be possible) to create an approach that encompasses everything, and it is for this reason that a plethora of generative techniques exists. In order to address project- and domain-specific needs, it is necessary to adjust to conditions in practice. The novelty of the generative technique presented in this work (see Chapter 6) is based on such circumstances. This technique fills a gap by addressing UML2 activities – something that has not been done as yet.

In contrast to the findings of Nicolás and Toval, the manner in which research works are validated has shifted in favor of industrial case studies. Of the works listed in Table 2.2, 16 were validated at least in part through industrial case studies. Despite the (often promising) validation in an industrial context, the application of these approaches in practice still remains an exception. The gap in industrial practice is tied to the aforementioned domain, project and or-

ganizational specifics. Of all the works in Table 2.2 only two claim productive use in industry: that of van Lamsweerde [130] and of Robinson-Mallett [200]. The work of van Lamsweerde features *Objectiver*, which is a commercially available tool. This tool supports an entire goal-oriented methodology that also allows for the generation of textual requirements from models. However, it requires that the ideas of goal-oriented RE be followed – a practice that has recently been shown to have limited impact within real-world settings [149]. Robinson-Mallett claims that his approach was applied in more than ten specification projects in industry, but gives no details on the contextual circumstances or results. Aside from the manner in which the proposals are validated, all works justify or at least motivate their necessity by emphasizing the importance of the examined modeling language/diagram type in industrial practice. The generative technique that is part of this work follows this idea, but it derives its relevance from the fact that the need for such a technique has been voiced by practitioners and from the lack of a general-purpose solution to address this need.

#### 2.4.2 Requirements-to-Model Approaches

Requirements-to-model approaches use requirements or some form of specification in a textual form to extract various types of models from a wide variety of modeling languages. In literature, two main purposes of requirements-to-model approaches are mentioned. On one hand, they aim to facilitate requirements analysis by providing a graphical representation and aid requirements validation by converting textual requirements into what is assumed to be a more formal representation than the original text [199, 207]. On the other hand, requirements-to-model approaches are recognized from an MDD point of view [228]. As such, a development effort consists of a sequence of model transformations where the result is deployable source code [243]. In that sense, requirements-to-model approaches are the first transformation of the sequence. At the same time, these approaches differ from *classic* model transformations, which are essentially transformations of directed graphs [154]. Requirements-to-model approaches usually make use of NLP [178] or techniques that fall within the realm of artificial intelligence [5]. Although the generation of models from requirements represents a valid form of co-existence between requirements and models, this aspect is only of marginal importance in this thesis. The methodology studied in this thesis does not follow the aforementioned aims of requirements-to-model approaches and these approaches are therefore merely presented for the sake of completeness.

An overview of requirements-to-model approaches is shown in Table 2.3. Note that only works that explicitly mention requirements or

Table 2.3: Works on requirements-to-model approaches  
(NL (Unstructured NL), UC (Use Cases), US (User Stories))

Target Model	Origin	Author
Activity Diagram	NL	Ilieva and Ormandjieva [116], Fliedl et al. [85]
	UC	Śmiałek et al. [216], Tiwari et al. [228]
BPMN	NL	Friedrich et al. [89], Honkisz et al. [112]
Class Diagram	NL	Capuchino et al. [46], Overmyer et al. [179], Insfrán et al. [117], Ambriola and Gervasi [11], Fliedl et al. [85], Debnath et al. [63], Letsholo et al. [136], Ahmed et al. [7]
	UC	Subramaniam et al. [224], Yue et al. [244]
Conceptual Models	US	Robeer et al. [199], Lucassen et al. [140]
Feature Models	NL	Niu and Easterbrook [166], Ferrari et al. [80], Itzik and Reinhartz-Berger [122]
	UC	Feijs [78]
Message Sequence Charts	NL	Kof [128]
	UC	Yue et al. [244]
Sequence Diagram	NL	Insfrán et al. [117]
	UC	Diaz [121], Śmiałek et al. [216], Yue et al. [244], De Souza et al. [62]
State Machines	NL	Kof [129]
	UC	Somé [218], Yue et al. [242], Scandurra et al. [207]
Use Case Diagrams	NL	Ilieva and Ormandjieva [116], Ambriola and Gervasi [11], Santos et al. [206]
	US	Elallaoui et al. [73]
Other		
Data Types	NL	Abbot [4]
Object Diagrams	NL	Mich [157]
Architecture Concepts	NL	Grünbacher et al. [101]
Entity-Relationship	NL	Ambriola and Gervasi [11]
Live Sequence Charts	NL	Gordon and Harel [93]

requirements documents as a possible source of text are considered. In the first column, the table lists which model types are created from the requirements. The second column lists the type of text that is used to derive the model from. There is a distinction made between NL text without an underlying structure and text with an underlying structure, such as use cases and in more recent works, user stories. The third column lists the author(s) for each combination of target model and the type of text source. Some works appear multiple times in the table. These approaches encompass techniques from requirements to multiple different model types and are accordingly listed for each target model.

Most works generate class diagrams, which is not surprising as this is one of the most widely used diagram types [138]. Nevertheless, this result might have been different if text sources other than requirements or requirements documents had also been included [145]. Regarding the origin, most works attempt to use unstructured NL as a source text. This is an ambitious endeavor because of the high level of freedom unstructured NL text provides as well as its inherent ambiguity (see Chapter 1). However, since most text comes in this form [31], it can be considered the most relevant. Use cases also play an important role in RE and are therefore a widely used form of specification [228]. The same is true for user stories, which have become increasingly important as a result of the wide adoption of agile development methods in practice [125, 139].

Although, there are many requirements-to-model approaches, they lack relevance in practice [5, 243], because they exhibit poor accuracy. These approaches thus require human involvement in post-editing even though they are supposed to be fully automated [140]. Yet, full automation does not seem to be within reach in the near future [29, 203].

#### 2.4.3 *Bridging the Gap between Graphical Models and Textual Requirements*

Employing models in the development of software systems makes it necessary to establish a comprehensible connection between model elements and requirements [16]. This necessity arises independently of how the models are used and relate to the requirements [245].

As described in Section 2.1, this connection is realized through a capability called *traceability*. As with "pure" requirements traceability, it is necessary to relate corresponding representations (graphical models and text) with one another to avoid omissions and prevent inconsistencies [2, 238], which might otherwise complicate the use of a specification approach that uses coexisting software artifacts. Accordingly, traceability between the elements of a graphical model and the coexisting textual objects must be ensured.

Traceability between requirements and all types of models has been a focus for the research community for quite some time [238]. One challenge associated with traceability between heterogeneous artifacts is the use of different tools to manage these artifacts. As a wide variety of tools exist for both RE and MDD tasks, not all combinations are predictable. Nevertheless, to ensure bidirectional traceability (i.d., traceability both forwards and backwards) the tools on the RE as well as the MDD side must accommodate traceability information.

A practical solution to realize trace links between different tools is Open Services for Lifecycle Collaboration (OSLC) [176]. OSLC defines a set of specifications for integrating different software development tools with each other. While OSLC offers capabilities beyond simply realizing trace links, it is nonetheless still suitable for this purpose. However, OSLC requires that each tool vendor implements an OSLC interface in its tool. This implementation might not exist for tools that are not designed for the purposes of RE and MDD (e.g., Microsoft Word/PowerPoint). This practice, of using tools not designed for specialized purposes, is not uncommon in industry [57, 222]. In addition, the data might not be managed in tools, but rather in interchange formats, such as ReqIF for requirements [173]. As a result, to support a wide range of possibilities the state of the art is to associate the entities of one representation with some type of evidence of the relating entity (e.g., an ID) [159].

## 2.5 SCOPE OF THIS THESIS

With regard to MDD, there have been a number of efforts to determine whether MDD fulfills its promises of facilitating software development in practice [222]. It has been found that the challenges of adopting MDD in an industrial environment are not only related to technical issues but are often more social in nature [236]. This gap between research efforts (in-vitro) and realization in industry (in-vivo) not only affects MDD, but also almost all newly introduced and sophisticated ideas – an issue that has been noted with regard to approaches in RE [149] as well. As a result, the application of MDD practices for RE purposes must also be considered in a real-world environment to assess the impact of such applications. In more general terms Orlikowski notes [177]:

“Technology per se can’t increase or decrease the productivity of workers’ performance, only use of it can.”

Hence, every approach developed in research has to prove its applicability in practice. Considering the related work in this chapter, out of the numerous approaches validated with the help of practitioners, only van Lamsweerde [130] and Robinson-Mallett [200] actually claim a productive use for their approaches in industry. However, it

cannot be assumed that their ideas are applicable in every situation. Thus, it is necessary to assess methodologies created and used by the practitioners themselves.

The scope of this thesis includes just such an industrial implementation of a specification approach. An approach that is not only applied in an industrial setting, but that is also developed by practitioners to fit their own needs with consideration of their day-to-day work and organizational environment – circumstances that still have received only little attention from the RE community [10]. Unlike other approaches, the stakeholders involved in the examined approach are not merely subjects in a research effort but the main (and sole) actors. From **Problem Statement 1**, one of the questions this thesis aims to answer is:

What are the implications (benefits, pitfalls, drawbacks) of a specification approach using coexisting graphical models and a textual representation implemented by practitioners independently?

Answering this question aims to provide insights into the examined specification approach, which uses coexisting UML2 activity diagrams and structured text for representations of requirements. Since practical application causes a number of issues itself, solutions to these issues are also developed. As the contributions in Section 1.4 and **Problem Statement 2** have already implied, inconsistencies between artifacts present significant risks in practice. Considering this, we aim to answer the question:

How can inconsistencies between the activity diagrams and the textual representation be addressed?

The answer to this question should guarantee the alignment of the coexisting activity diagrams and the textual representation. As a result, the quality of the artifacts should be improved, which positively affects user acceptance.





## Part II

### ANALYSIS OF A SPECIFICATION APPROACH BASED ON COEXISTING ACTIVITY DIAGRAMS AND TEXTUAL REPRESENTATIONS



## COEXISTING GRAPHICAL AND STRUCTURED TEXTUAL REPRESENTATIONS OF REQUIREMENTS: INSIGHTS AND SUGGESTIONS

---

Published in *Requirements Engineering Foundation for Software Quality*.  
- LNCS (10753) (p. 265 - 280) [23].

### TERMS OF USE

The final authenticated version is available online at:  
[https://doi.org/10.1007/978-3-319-77243-1\\_16](https://doi.org/10.1007/978-3-319-77243-1_16)

### BROADER CONTEXT WITHIN THE THESIS

The application of a specification approach that uses graphical and textual representations of requirements and is implemented by practitioners independently has not yet been examined in an industrial context (see Chapter 2). In this chapter, a study is presented in which we analyze an approach that uses activity diagrams as graphical representation and hierarchically structured text as textual representation. How users work with the content generated by the approach and what challenges they face, is analyzed. This analytical portion of the study addresses **Problem Statement 1** and represents contribution I of the thesis.

In order to address the identified challenges, suggestions are made regarding how to improve the application of the approach. These suggestions help address **Problem Statement 2** and represent contribution III of this thesis.

### AUTHOR CONTRIBUTIONS

The author list includes Martin Beckmann, Christian Reuter, and Andreas Vogelsang. The author of this thesis was the lead author of the publication. He wrote the majority of the article and designed, conducted, and evaluated the interviews. Christian Reuter managed organizational issues at the industry partner and independently repeated the evaluation to confirm the results. Christian Reuter and Andreas Vogelsang both provided advice regarding the study design and contributed to the writing of the paper.

# Coexisting Graphical and Structured Textual Representations of Requirements: Insights and Suggestions

Martin Beckmann<sup>1</sup>, Christian Reuter<sup>2</sup>, and Andreas Vogelsang<sup>1</sup>

<sup>1</sup> Technische Universität Berlin, Germany

<sup>2</sup> Daimler AG, Germany

**Abstract.** [Context & motivation] Many requirements documents contain graphical and textual representations of requirements side-by-side. These representations may be complementary but oftentimes they are strongly related or even express the same content. [Question/problem] Since both representation may be used on their own, we want to find out why and how a combination of them is used in practice. In consequence, we want to know what advantages such an approach provides and whether challenges arise from the coexistence. [Principal ideas/results] To get more insights into how graphical and textual representations are used in requirements documents, we conducted eight interviews with stakeholders at Daimler. These stakeholders work on a system that is specified by tabular textual descriptions and UML activity diagrams. The results indicate that the different representations are associated with different activities. [Contribution] Our study provides insights into a possible implementation of a specification approach using mixed representations of requirements. We use these insights to make suggestions on how to apply the approach in a way that profits from its advantages and mitigates potential weaknesses. While we draw our conclusions from a single use case, some aspects might be applicable in general.

**Keywords:** Model-Driven Software Specification; Graphical Models; Requirements Documents; UML Activity Diagram

## 1 Introduction

Eliciting and specifying requirements by means of models is becoming more and more popular in the development of complex embedded systems [1]. However, these models usually accompany and complement textual requirements and do not replace them. Therefore, many requirements documents contain graphical and textual representations of requirements side-by-side. This combined use of graphical diagrams and textual descriptions is considered beneficial for the requirements management process [2, 3].

In practice, there are more substantial reasons why the same information may be expressed in a graphical model and also in an accompanying text. For

example, industrial applications, tool support, and model exchange for graphical models are still not standardized [4] and, as a result, manufacturer/supplier handover is still performed by textual documents. This is especially important, since these textual documents often serve as the basis for legal considerations between the contractors [3, 5]. Also, due to different backgrounds of the stakeholders, not everyone is capable of understanding the graphical models [6].

Maintaining and updating information in graphical and textual representations is often performed manually. In previous work, we have shown that this is a potential source for inconsistencies and quality issues in the requirements specifications [7]. Moreover, best practices and guidelines for when and how to use graphical or textual representations are missing. This leads to discussions about the validity of the representations, when deviating representations exist.

Without a deeper understanding of how the different representations are used and why they coexist, it is hard to come up with measures for ensuring consistency or to decide how content should be represented. Therefore, we are interested in how coexisting graphical and textual representations of requirements are used by stakeholders of the system. For this purpose we considered one particular instance of this case in practice, where a team at Daimler uses UML Activity Diagrams to provide a high-level overview of the activation conditions for a vehicle function. The information contained in this model is afterwards transferred into a tabular textual representation that is then further detailed.

We conducted eight interviews with practitioners at Daimler. Three interviewees have developed the specification approach described above. Five interviewees work with the resulting requirements document. From these interviews, we derive a model that describes for which activities stakeholders use graphical or textual representations. Also, we use the acquired data to provide suggestions on how graphical and textual representations should be used to leverage their potential and avoid pitfalls which would lead to quality issues.

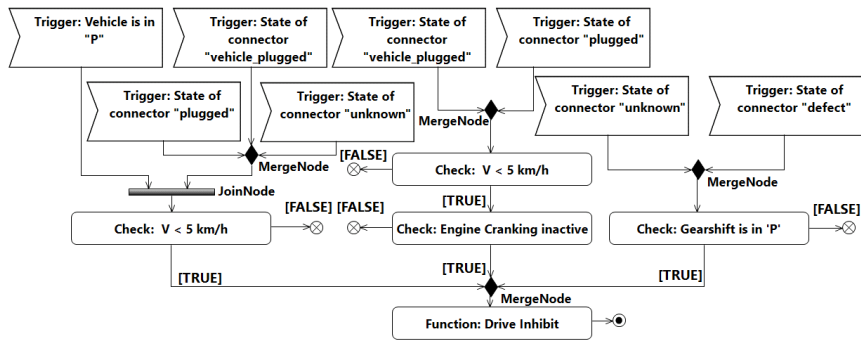


Fig. 1: Activity diagram of the function *Drive Inhibit*

## 2 Background

A team at Daimler employs UML activity diagrams [8] to specify functions of a system. The diagrams are used to get an early overview of the desired function behavior with a special focus on the activation of the function, execution conditions, functional paths, and deactivation. Fig. 1 depicts a diagram of the system. The actual behavior of the activated function is described in the *Action* node labeled with *Drive Inhibit* (bottom of the diagram). The activation of the function is described by a combination of triggers and checks for conditions. This pattern to describe functions is also known for building textual requirements [9]. Activity diagrams are interpreted according to the requirements-level semantics of activities as defined by Eshuis and Wieringa [10]. As such, we assume that each node executes as soon as a token is placed on that node (by a transition or by occurrence of events). We also assume that the time required to execute a node is infinitely short. Control nodes have the usual semantics: *MergeNodes* (diamonds) and *JoinNodes* (bars) represent OR connections and AND connections, respectively. All the activity diagrams of the system are modeled in a similar way in regard to the used pattern, structure and layout.

The activity diagrams are then embedded in a textual requirements specification in two representations: (a) graphically as an image, (b) in a tabular, textual form which is supposed to reflect the same behavior as the activity diagram. The tabular representations may be refined and extended later.

Fig. 2 shows the textual representation of the activity diagram in Fig. 1 as we found it in the specification document of our industry partner. The basic idea of the textual representation is to represent the triggers and checking conditions which govern the execution of a function as a kind of AND-OR table with postfix boolean operators. As such, the textual representation emphasizes the propositional logic aspect of the behavior. Each row represents an object, which is described by a set of attributes (columns). These attributes are needed to display the relevant information of the activity diagram in the requirements document. The *ID* attribute contains a unique identifier of the object. The *Text* attribute is a textual description of the object and is supposed to be equal to the text of the corresponding element in the activity diagram. It also contains the boolean operators which connect multiple elements within a cell or connect one row to the next row on the same *Level*. The *Level* is an attribute to structure the objects hierarchically. It is derived from the structure of the activity diagram. The *Type* attribute denotes whether an object is a function, a trigger or a condition to be checked. The object types in the table are derived from the types of the corresponding elements in the activity diagram.

Note that the activity diagram and the textual representation exhibit a number of differences with respect to both placement of elements and the specified behavior. E.g., the element *Check: Engine Cranking inactive* has the predecessor *Check:  $V < 5 \text{ km/h}$*  in the activity diagram, while in the textual representation the element *Vehicle Gear selector is in position "P"* is the predecessor. Besides, some rows in the textual representation mistakenly have a connector at their end (*ID 1113, 1233*), although there are no further rows on the same *Level*.

These issues may originate from the manual generation of the textual representation and changes over time. We have addressed these problems in a previous paper [7].

ID	Text	Level	Type
1000	<b>1.1.1.1.1 Drive Inhibit</b>	6	Function
1236	State of connector "unknown" OR State of connector "defect" OR	7	Trigger
1237	Vehicle Gear selector is in position "P" AND	8	Check
1113	Engine Cranking inactive OR	8	Check
1111	State of connector "plugged on vehicle side" ("VEH_PLUGGED") OR "plugged on vehicle and EVSE side" ("PLUGGED") OR	7	Trigger
1112	Vehicle velocity is below 5 km/h	8	Check
1114	Vehicle Gear selector is in position "P" OR	7	Trigger
1232	Vehicle velocity is below 5 km/h	8	Check
1233	State of connector "plugged on vehicle side" OR State of connector "plugged on vehicle and EVSE side" OR State of connector "unknown" AND	7	Trigger
1238	Vehicle velocity is below 5 km/h	8	Check

Fig. 2: Textual representation of the function *Drive Inhibit*

The sample in Fig. 2 only depicts the contents derived from the activity. Besides the mentioned attributes, the document may contain other attributes used for further development. Also, the textual document may contain more detailed information in the form of further requirements and descriptions. These entries may be both formal (e.g., parameter values) and in freely-written natural language.

### 3 Related Work

Graphical notations as a means to ease the understanding of complex systems have been used in different contexts [11, 12]. Nevertheless, despite showing several advantages there are drawbacks such as end users' unfamiliarity with graphical notations and limits on the displayable details in visualizations. Moreover in requirements engineering, research has identified the need for different representations of requirements [13]. A possibility to tackle these issues is to use accompanying text for graphical models. Arlow et. al. introduced an approach called *Literate Modelling* that works with this idea and employs UML models as the graphical models [6]. This concept of coexisting graphical models and textual descriptions was picked up and discussed for future tools in requirements engineering [14]. In addition the approach is supported by ideas using a graphi-

cal model as a basis to generate a structure for requirements documents and requirements itself [15].

However, to the best of our knowledge, there is only a small number of works on the topic of how to apply the approach and on its impact. Aside from computer science, it has been shown that the combined use of words (written and spoken) and pictures has a beneficial effect on a person's perception [16]. Still, it is also known that readers focus on the representation that takes the least effort to understand, in case they contain the same information [17].

A study of Burton-Jones et. al. with student participants investigates whether a combination of representations is beneficial [18]. They report a positive impact for understanding a new system by using conceptual graphical models and a textual narrative, but do not give details on how to implement such an approach in practice. Our intent is to improve the understanding in this area by interviewing practitioners and to make suggestions on how to implement such a mixed representation approach in the best way possible.

## 4 Study Design

To gain a better understanding of how the approach is used and how the involved parties work with the activity diagrams and the textual parts, we conducted an interview study with stakeholders of one particular system. We designed the study along the recommendations of Runeson and Höst [19].

**Research Objective:** We want to know how the different stakeholders use the graphical models and the textual descriptions, how and where they make changes, and how they ensure consistency of the specification. Additionally, we are interested in the stakeholder's perception of advantages, challenges, and best practices of the application of the approach.

To reach this objective, we pursue three research questions (RQ):

**RQ1: For which activities do the stakeholders use which representation?** With this research question, we aim at getting insights about the use of different representations in order to be able to derive suggestions for working in a setting with coexisting representations.

**RQ2: What are the reasons why stakeholders use one or the other representation for specific tasks?** We want to find out why stakeholders use one of the representations for certain tasks. This is meant to provide insights on the benefits the graphical models offer and how the coexisting artifacts are used in the work of the involved persons.

**RQ3: What challenges arise in the combined use of graphical models and text and how should they be addressed?** We want to know what problems the stakeholders face. This gives us an idea on potentials for improvement. Also, this RQ is used to derive suggestions for the use of graphical models in combination with text for specifying functions.

**Study Object:** We conducted this study in the context of the development of one particular system. The system contains functions involved with charging the batteries of Plug-in Hybrid Electric Vehicles and Battery Electric Vehicles.



As such, the system contains requirements that are relevant for safety as well as for usability. Overall, there are 14 functions in the system which are described by the approach mentioned in section 2. These functions contain a total of 22 activity diagrams and almost 2,000 objects (including requirements, descriptions and headings). The additional activity diagrams result from the fact that some subfunctions of the functions are also described by activity diagrams and text.

**Data Collection:** We conducted interviews with eight stakeholders of one particular system. The majority of the interviewed stakeholders (five) either depend on the contents of the requirements document directly or on content which is derived thereof automatically or manually. The rest of the stakeholders (three) are concerned with the methods that are applied to specify systems and components at Daimler. We group the participants into three groups: those involved with the testing of the functions (in the following referred to by:  $T_1$ ,  $T_2$ ), those who use the specified functions to specify components ( $C_1$ ,  $C_2$ ,  $C_3$ ), and those developing the applied methods ( $M_1$ ,  $M_2$ ,  $M_3$ ).

The interviews were performed by following an interview guideline. The interview guideline was created in multiple iterations. In each iteration the structure and questions were refined by discussions with other researchers and practitioners of our industry partner to ensure that the research questions are properly addressed. However, the interviews were conducted as open interviews. In case the participants mentioned issues aside from the questions of the guideline, we did not interrupt and followed up on these issues in some cases. Also, insights gained during the interviews were considered in the following interviews.

The first part of each interview concerned the background of the interviewee. We asked questions on how long they have been working with the contents of the system, what their current role is, whether there was prior knowledge in dealing with graphical models, and what their general attitude is towards the use of graphical models.

The second part aimed at eliciting facts about their work. This question covered what the participants actually use the activity or text for as well as in what way the two artifacts provide different information for their tasks. Furthermore, we asked what purposes the activity diagram and the textual description respectively fulfill. As the participants  $M_1$ ,  $M_2$  and  $M_3$  do not directly work on the contents we engaged them in a discussion about their idea how the artifacts are supposed to be used. In addition, we asked the participants for their general impression on the quality of the activity diagrams and the accompanying text.

The third part aimed at initiating a discussion with the participants. We wanted to know where they see advantages in the current approach, what challenges they face in applying it in their own work and how to possibly deal with them. We also wanted to find out how they perceive the influence of the approach on the contents they are provided with. Hence, we encouraged the participants to give their opinion on the way the system's functions are specified and what consequences they expect for their tasks. Furthermore, we wanted to find out whether they can imagine a different process for the specification of functions and how that would differ from the current approach.

The majority of the interviews (five) was conducted on site. The rest of the interviews (three) was conducted by telephone. We ensured that the statements of the participants were handled in an anonymous way to guarantee honest answers. The interviews were scheduled to last about an hour. In the end the shortest interview lasted 32 minutes, while the longest took almost 90 minutes. The interviews were recorded.

**Data Analysis:** The first author created transcripts of the interviews. These transcripts summarize the whole interview and contain the essential statements of the participants. Due to the open nature of the interviews the number of statements differ from participant to participant. We analyzed the transcripts by applying qualitative coding [20]. The analysis was performed by the first and the second author. Our first step was to read the interview transcripts to get an overall impression. This impression was used to extract a first set of concepts. These concepts were then discussed in regard to their relevance towards the research questions. The discussion resulted in a common set of concepts. We then checked the transcripts for information, which fit the identified concepts. This task was performed independently and afterwards the coding was compared. In case of deviations the results were discussed until we reached a mutual agreement. This mutual agreement led to the omission of a number of statements, since they did not directly address the research questions. It turned out that some of these omitted statements covered interesting aspects nonetheless. Hence, it was decided to repeat the process in the same manner with additional concepts in order to include these aspects. We deduced the relevance of these aspects by the fact that they were mentioned by multiple participants.

## 5 Study Results

### 5.1 Demographics & Background

The interviewed participants have been working for our industry partner for a time period between 2 and 28 years. All of the participants stated to have prior experience in working with graphical models. This encompassed statements between some familiarity with UML and similar graphical notations to expert knowledge in the application of graphical models in the development of systems. Also, all participants stated to have a positive attitude towards the use of graphical models. Those statements ranged between seeing minor benefits to the impression that graphical models are nowadays necessary to be able to comply with standards and to create high-quality requirements.

### 5.2 Benefits & Use of the Approach

To address RQ1 and RQ2, we considered the answers to the questions that concerned the activities the participants perform during their work as well as parts of the discussion revolving around the advantages they perceive.

The tasks the participants perform are shown in Fig. 3. Boxes denote activities, while ovals represent artifacts. The lines show the associations that the

participants mentioned in the interviews. The arrow between the two artifacts indicates that the graphical model is the initial artifact which is used to derive the textual descriptions.

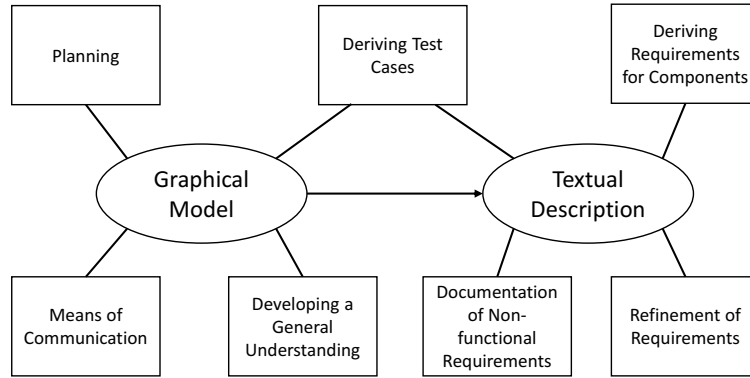


Fig. 3: Tasks associated with the artifacts

To use the graphical models as a means of communication and to develop a general understanding was identified as a task by almost all participants. Additionally, two participants ( $M_1$ ,  $M_3$ ) mentioned to use the graphical model during release planning. They use the relations between the elements of the diagram to gain insights into dependencies between underlying components, which in turn facilitates the planning. The only task associated with both representations is deriving test cases. In this matter, participant  $T_2$  explicitly mentioned that the activity diagrams are the actual basis to create some of the test cases and not just a supporting alternative view of the text.

Nevertheless, the groups involved in testing and those responsible for components of the system both stated to rely mostly or even solely on the textual description to derive their own artifacts (test cases and components requirements). Furthermore the textual description was mentioned to be used to refine requirements and to provide more details on contexts and surrounding circumstances by all of the participants.

Aside from the performed activities, there seems to be confusion about the use of the approach itself. There was no common understanding between the participants on whether the textual or the graphical representation should be created first, which one is used in case of inconsistencies, and where changes are incorporated. Different statements were made on this topic. Some participants mentioned that they are unaware of how the artifacts are created and where to incorporate changes.

Moreover, the answers of the participants offered insights on what they think the artifacts are used for and what benefits the approach offers. Table 1 and Table 2 show an overview of all statements the participants made about graphical

models and textual descriptions, respectively. A ✓ denotes that the participant made that statement while a — denotes that the participants did not make mention of that fact.

Since all participants mentioned to have a positive attitude towards the use of graphical models, it is not surprising that their use is considered beneficial. Many even mentioned that they consider the use of graphical models as a necessity. As the associated tasks have shown, there is a lot of agreement that activity diagrams are used as a means of communication and a basis for discussion. Also, it was mentioned explicitly by almost all participants that the diagram improves the general understanding of a function.

For the textual descriptions, most participants mentioned that they see the text as the reference and it is used to provide details. The fact that the text is necessary because of legal considerations was only mentioned explicitly by participant  $T_2$ . The necessity to support stakeholders who are unfamiliar with the use of graphical models was stated by  $C_1$ ,  $T_2$  and  $M_2$ .

### 5.3 Challenges & Possible Improvements

To answer RQ3, we asked how they perceive the quality of the activity diagrams and their textual representation. More specifically, we wanted to know how they like the way the artifacts are structured and whether they face challenges by maintaining coexisting artifacts.

All participants emphasized that consistency is a major problem in the way the approach is currently applied. As a consequence, all participants would appreciate automatic support for deriving the textual description from the activity diagrams. They assume that this would have a positive impact on their work.

The textual representation was criticized with regard to its interpretation. Some participants said that they would prefer a different structure as the current one is not intuitively understandable. However, further inquiries on this issue revealed that the boolean operators without following rows on the same level (described in section 2) are not perceived as a problem.

Many issues with the activity diagrams were mentioned. For instance, critique was expressed on the depiction of the activity diagrams. This critique focused most often on the fact that the diagrams are not uniformly designed using the same tool. Also, the pattern depicted in Fig. 1 is not strictly enforced. Furthermore, the contained information was criticized in regard to both the amount and level of detail. This point encompassed different opinions of the participants. Some of them stated that required information, such as signal names and values, are missing in the diagrams. Others stated that there are too many elements and details in some diagrams to understand a function properly. Yet, others said that the activity diagrams contain information (e.g., of other components) that is not relevant for them.

As the layout of a graphical model has a major impact on its understandability [21], we also wanted an opinion on the quality of the layout. All of the participants mentioned to be satisfied with the quality in that regard. Still, the way the activity diagrams are embedded in the tool was criticized. The diagram

Table 1: Statements about the use of graphical models by participants

Participant	considered beneficial	considered necessary	means of communication / discussion	improves under- standability	should be basis for text	display architecture	represents relations	used for planning
$C_1$	✓	—	✓	✓	✓	✓	✓	—
$C_2$	✓	—	✓	✓	✓	—	✓	—
$C_3$	✓	—	✓	✓	✓	✓	✓	—
$T_1$	✓	—	✓	✓	✓	—	✓	—
$T_2$	—	✓	—	—	—	✓	—	—
$M_1$	—	✓	—	✓	✓	✓	✓	✓
$M_2$	—	✓	✓	✓	✓	✓	✓	—
$M_3$	—	✓	✓	✓	—	—	✓	✓

Table 2: Statements about textual descriptions by participants

Participant	acts as reference	legal consid- erations	contains details	handover for supplier	used for non-functional requirements	support stakeholders unfamiliar with models
$C_1$	✓	—	✓	—	—	✓
$C_2$	✓	—	✓	—	—	—
$C_3$	✓	—	✓	✓	—	—
$T_1$	✓	—	—	—	—	—
$T_2$	—	✓	✓	✓	—	✓
$M_1$	—	—	✓	✓	—	—
$M_2$	—	—	✓	✓	✓	✓
$M_3$	—	—	✓	—	—	✓

is included as a picture in a cell in the requirements document. Since the default size of such a cell does not allow for the display of the complete diagram, it is necessary to adjust its size manually in order to see the full diagram.

#### 5.4 Beyond the Research Questions

Since we designed the study as an open interview, many things were mentioned that did not directly address our research questions. Still, some of these statements are within the scope of our research objective.

Regarding the question what the graphical model is used for, the answer that appeared most often was an improved understandability. Further questions in that matter revealed that the understanding concerns mostly relations between the elements in the graphical representation. Aspects of activities such as independent executability of actions and asynchronous behavior were never mentioned. When we specifically asked for that, it was stated, that this is of no importance on that level of description.

As the automated generation of the textual description from the graphical models was mentioned, we wanted to know whether the capability of synchronization of the graphical and textual representation is needed. The participants answered that this capability would be nice-to-have, but all agreed that changes are best incorporated in the graphical model.  $M_1$ ,  $C_2$  and  $T_2$  said, it should not be possible to change aspects of the graphical model in its textual description and hence a synchronization in the backwards direction should not be allowed.

Towards the end of the interviews, we challenged the approach as a whole and asked whether they could work without the textual representation. Because of the already mentioned uses of the text, about half the participants instantly stated that it does not seem possible. The rest was open to the idea, but had doubts, because of organizational considerations (e.g., handover to suppliers, legal issues) and also stated the necessary models would mitigate their main advantage — the capability of offering a clear overview. Participant  $T_2$  said this would require major modifications in the company structure. It would be possible if all development tasks from suppliers are reintegrated to one place.

## 6 Discussion

### 6.1 Findings from our Study

All in all, there seems to be a common understanding between the different stakeholders on why they use this approach and on what to use each artifact for. We derive this conclusion from the fact that all of the stakeholders consider the two coexisting artifacts to be at least beneficial. This is also reflected by the fact that there is a high-level of agreement towards the way the respective artifacts are used. Furthermore, the association of specific tasks with certain artifacts indicates that both the graphical representation and the textual representation are necessary to manage the complexity of today's systems and hence create high-quality requirements specifications.

The graphical representation is mainly seen as a means of communication and discussion and for improved understandability by almost all participants. Communication and discussions are necessary to make sure the behavior is as originally intended. A proper understanding of the function is mandatory for the stakeholders. These two purposes facilitate subsequent tasks such as deriving requirements for components and the manual generation of test cases. Thus, we see the diagram in a rather supportive role. These results also indicate that the graphic models are primarily used for the purpose of visualization and not for expressing precise semantics. In consequence it serves a wallpaper use [22].

The only aspect that was commented conflictingly about the graphical models regarded their depiction. Participant  $T_1$  mentioned, that she would rather prefer more elements in a diagram than scrolling to a different diagram to get more information. Participant  $T_2$  mentioned that the maximum number of elements in a diagram should be restricted to about seven elements and, if further elements are required, they should be nested into a linked diagram. In addition, some participants complained about information in the diagrams that is not relevant to them. This conflict cannot be resolved by using a single graphical representation of a function for all stakeholders (cf. [13]).

As for the textual representation, the results strongly suggest that it is in fact the preferable medium to accommodate refinements and details. Half of the participants mentioned the need to support stakeholders unfamiliar with graphic models. This is an issue that constantly appears in contexts where models are used. The coexistence of textual descriptions and graphical models appears to be a possible solution to this issue [23]. Nevertheless, there might be more fitting possibilities to arrange the textual representation than the one currently used (see [24] for a study on different textual representations of activity diagrams).

Although the graphical representation is created as a first step for the specification, its use is not restricted to the specification phase. As our participants perform a variety of tasks, we found out that the graphical model fulfills more purposes than just being a starting point for further specification. Amongst others it is used to derive test cases and to support understanding of the intended behavior. Hence, it proved to have been a good idea to consider participants outside the group of people who create the graphical models and textual descriptions. This selection of participants, on the other hand, also explains the lack of understanding which artifact is created at which step in the process, where changes are incorporated, and which artifact has to be used in case of inconsistencies. In hindsight, it turned out that the lack of a definition which artifact is used as the lead is also linked to the study object. Although half of the participants mentioned that the text is used as a handover and for legal considerations, this mainly applies to the derived component specifications. System specifications are mainly used internally and hence using the textual representation as the reference is not strictly enforced.

With regard to these insights we conclude that in our case using a textual and graphical representation on the same level of abstraction is an appropriate means in the development of systems since the artifacts serve different purposes.

To make the most of the approach, we make suggestions that aim at mitigating the found weaknesses and taking advantages of the identified strengths.

## 6.2 Suggestions

Based on the insights we make suggestions on how to implement a mixed representations approach in order to leverage the potentials of the respective representations. From the high level of agreement concerning that the activity diagram should be used as a basis for the text, we conclude that the activity diagram is indeed an adequate starting point for the specification process of our industry partner. This finding is largely in line with research on the use of graphical models that emphasizes its use during the early stages of development [25]. Hence, this section starts with suggestions on the use of the activity diagrams and proceeds with suggestions on the textual representation of our industry partner.

**Use of the Activity Diagrams.** One of the major factors to the success of graphical models is that it needs to be understood by as many stakeholders as possible. To achieve this, it is paramount to design the models according to a defined pattern. Also, we recommend to use a common tool for the modeling in order to ensure a uniform look, although this might be hard to enforce. Nevertheless, access to the tool should be granted to all who make use of the activity diagram. This is required to address the problem with the handling of the diagram. From the different opinions on the contained information, we conclude that a mechanism is needed to tailor the models according to each individual's needs. This suggestion has been stated before [13] and is in line with established solutions on using textual requirements [26].

**Use of the Text.** Deriving the text from the activity diagram avoids inconsistencies and hence ensures that the same behavior is described by both representations. Aside from the situation of our industry partner, there are already a number of approaches dealing with the generation of requirements specifications (or parts thereof) from models [15]. Following our participants the text can be used to incorporate refinements and details. As the complementary information may also be freely written in natural language, this representation may in fact be better suited for stakeholders unfamiliar with the notations of activities. Detailed information should only appear in the text to avoid further consistency issues and to guarantee the main purpose of the activity diagram is not impaired — to maintain a high-level overview.

**Incorporation of Changes.** As the appearance of changes is inevitable in the course of development, their incorporation in the artifacts must be considered. Changes to the relations of entities are easier to implement in the diagram. For textual changes it does not make much difference which representation is used. Nevertheless, to avoid inconsistencies only a single artifact should be used.



Hence, the activity diagram should accommodate changes which affect both representations, although this might be hard to realize considering the fact that multiple persons work with the specification artifacts. The changes in the activity diagram are then propagated to the textual representation. It has to be noted that the additional textual content is not deleted or modified in the process.

Alternatively, changes could be automatically incorporated by using tools such as Projectional Editors, which automatically edit different projections of a common underlying model, in this case the activity diagram and its textual representation. However, this approach requires substantial efforts and accordingly trained developers [27]. Hence, a custom-made and lightweight solution to generate and update the textual representation might be better suited for the situation of our industry partner.

**Further Related Tasks.** As for the tasks of the respective artifacts, the situation displayed in Fig. 3 is already a good way of applying the strengths of the model and the text. The main concern of the graphical model is human-based analysis and the exchange of ideas between stakeholders. As such, the tasks of planning, improving understanding, and facilitating communication are prone to involve a visualization. Still, since the graphical representation provides a high-level overview, these tasks are restricted to early stages of development, when the required descriptions do not need to be detailed. Nonetheless, the defined syntax and semantics of a graphical model can also be used to automatically derive test cases [28].

### 6.3 Threats to Validity

The participating stakeholders were selected by the second author who is also actively participating in the development of the examined system. We did not follow specific selection criteria, except that participants must work actively on the examined system. However, the group of study participants only represent a subset of all people working actively with the requirements documents.

Furthermore we only had access to internal participants within one company. However, the activity diagrams and their textual descriptions must also be read and understood outside the company, such as legal authorities and suppliers. Their opinion is critical since inquiries on unclear issues require more effort between multiple organizations than inside a single company.

Also, our study examined the present situation of an approach using activity diagrams. The use of other graphical models might influence the proposed suggestions as well as the benefits and weaknesses we identified.

To answer our research questions, we only had access to a limited number of participants who actively work with this approach or are responsible for the applied methods. Also, we only gained insights into a single implementation of a mixed representation approach which uses activity diagrams and a very specific kind of textual representation. In conclusion, although our findings turned out to be consistent, our results can only be seen as a first step. Hence, further research is required to generalize our findings.

## 7 Conclusion and Future Work

In this paper, we present the results of a number of interviews we conducted to gain a better understanding of a specification approach that uses coexisting activity diagrams and tabular textual descriptions. The results incorporate an assessment of our participants on which artifact is suitable for which task as well as their opinion on the benefits of the respective artifacts. The use of graphical models for themselves as well as their use in coexistence with textual description on the same level of abstraction is perceived as beneficial. We use the insights gained by these results to derive suggestions. The suggestions serve the purpose of providing a guideline on how to implement such an approach in order to avoid inconsistencies and leverage its full potential.

Although we think that our results can be generally applied to approaches using coexisting graphical and textual artifacts, the results should be further validated by repeating the study with differing implementations of the approach. The differences might concern the type of graphical model and the pattern for textual description. Also, the extent to which practitioners benefit from our suggestions needs to be further examined. Moreover, the graphical and textual representations described in this paper are not the only artifacts. To handle the complexity of today's systems, further diagrams and associated documents might be needed. Ensuring the propagation of necessary changes to these artifacts is still not implemented in an acceptable manner and hence needs further investigation.

## References

1. Broy, M.: Challenges in automotive software engineering. In: International Conference on Software Engineering. (2006)
2. Davis, A.M.: Just Enough Requirements Management: Where Software Development Meets Marketing. Dorset House Publishing Co., Inc. (2005)
3. Sikora, E., Tenbergen, B., Pohl, K.: Industry needs and research directions in requirements engineering for embedded systems. *Requirements Engineering* **17**(1) (Mar 2012)
4. Reuter, C.: Variant Management as a Cross-Sectional Approach for a Continuous Systems Engineering Environment. In: Grazer Symposium Virtual Vehicle. (2015)
5. Maiden, N.A.M., Manning, S., Jones, S., Greenwood, J.: Generating requirements from systems models using patterns: a case study. *Requirements Engineering* **10**(4) (Nov 2005)
6. Arlow, J., Emmerich, W., Quinn, J.: Literate Modelling Capturing Business Knowledge with the UML. In: International Conference on the Unified Modeling Language. (1998)
7. Beckmann, M., Vogelsang, A., Reuter, C.: A Case Study on a Specification Approach using Activity Diagrams in Requirements Documents. In: International Requirements Engineering Conference. (2017)
8. Object Management Group (OMG): OMG Unified Modeling Language (OMG UML), Version 2.5. <http://www.omg.org/spec/UML/2.5/> (2015)

9. Firesmith, D.: Generating Complete, Unambiguous, and Verifiable Requirements from Stories, Scenarios, and Use Cases. *Journal of Object Technology* **3** (11 2004)
10. Eshuis, R., Wieringa, R.: Tool Support for Verifying UML Activity Diagrams. *IEEE Transactions on Software Engineering* **30**(7) (2004)
11. Huff, A.S.: Mapping Strategic Thought. John Wiley & Sons Ltd (1990)
12. Pidd, M.: Tools for Thinking: Modelling in Management Science. 3rd edn. John Wiley & Sons Ltd (2009)
13. Gross, A., Doerr, J.: What You Need Is What You Get!: The Vision of View-Based Requirements Specifications. In: International Requirements Engineering Conference. (2012)
14. Finkelstein, A., Emmerich, W.: The future of requirements management tools. In: Information Systems in Public Administration and Law., Österreichische Computer Gesellschaft (2000)
15. Nicolás, J., Toval, A.: On the generation of requirements specifications from software engineering models: A systematic literature review. *Information and Software Technology* **51** (09 2009)
16. Mayer, R.E.: The Cambridge Handbook of Multimedia Learning. Cambridge University Press (2005)
17. Payne, J.W., Bettman, J.R., Johnson, E.J.: The Adaptive Decision Maker. Cambridge University Press (1993)
18. Burton-Jones, A., Meso, P.N.: The Effects of Decomposition Quality and Multiple Forms of Information on Novices' Understanding of a Domain from a Conceptual Model. *Journal of the Association for Information Systems* **9**(12) (2008)
19. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* **14**(2) (2009)
20. Adolph, S., Hall, W., Kruchten, P.: Using grounded theory to study the experience of software development. *Empirical Software Engineering* **16**(4) (Aug 2011)
21. Mendling, J., Reijers, H.A., Cardoso, J.: What makes process models understandable? In: Alonso G., Dadam P., Rosemann M. (eds) Business Process Management. BPM 2007. Lecture Notes in Computer Science, vol 4714 (2007)
22. Drusinsky, D.: From UML activity diagrams to specification requirements. In: International Conference on System of Systems Engineering. (2008)
23. van Oosterom, P., Lemmen, C., Ingvarsson, T., van der Molen, P., Ploeger, H., Quak, W., Stoter, J., Zevenbergen, J.: The core cadastral domain model. *Computers, Environment and Urban Systems* **30**(5) (2006)
24. Beckmann, M., Vogelsang, A.: What is a Good Textual Representation of Activity Diagrams in Requirements Documents? In: International Model-Driven Requirements Engineering Workshop. (2017)
25. Lindland, O.I., Sindre, G., Sølvberg, A.: Understanding quality in conceptual modeling. *IEEE Software* **11**(2) (March 1994)
26. Weber, M., Weisbrod, J.: Requirements Engineering in Automotive Development - Experiences and Challenges. In: Joint International Conference on Requirements Engineering. (2002)
27. Berger, T., Völter, M., Jensen, H.P., Dangprasert, T., Siegmund, J.: Efficiency of projectional editing: A controlled experiment. In: International Symposium on Foundations of Software Engineering (FSE). (2016)
28. Beckmann, M., Karbe, T., Vogelsang, A.: Information Extraction from High-Level Activity Diagrams to Support Development Tasks. In: International Conference on Model-Driven Engineering and Software Development. (2018)



## A CASE STUDY ON A SPECIFICATION APPROACH USING ACTIVITY DIAGRAMS IN REQUIREMENTS DOCUMENTS

---

Published in 2017 *IEEE 25th International Requirements Engineering Conference (RE)* (p. 245 - 254) [25].

### TERMS OF USE

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

### BROADER CONTEXT WITHIN THE THESIS

The study presented in Chapter 3 revealed that inconsistencies between the graphical and textual representation of requirements and other quality issues with representations have a negative impact on the users' work. The study in this chapter aims to improve the understanding of these deficiencies. This chapter presents this thesis' second analytical contribution, contribution II, and addresses **Problem Statement 1**. Since the analysis of the results also suggests possible solutions (contribution III), **Problem Statement 2** is also addressed.

As a part of the study, we define different categories of inconsistencies and other quality issues in an informal manner (see Appendix A for a formal definition of the categories and the constructs necessary (textual representation and the activities) to formally express the categories). The number of occurrences for each of the categories is collected for the examined system. Also, a certain number of stakeholders determine the severity of the categories by assessing samples of the categories from their system.

Although the system in this study is the same as in the previous chapter, the stated key figures (i.e., number of functions of the system) differ. This is due to the fact that the studies were not performed at the same time and the system was subject to changes as it is constantly under development.

#### AUTHOR CONTRIBUTIONS

The author list includes Martin Beckmann, Andreas Vogelsang, and Christian Reuter. The author of this thesis was the lead author of the publication. He wrote the majority of the article and conceived, designed, conducted, and evaluated the studies and surveys. The counting of deficiencies was independently repeated by Andreas Vogelsang to confirm the results. Christian Reuter distributed the surveys to the participants from the industry partner. Christian Reuter and Andreas Vogelsang both provided advice on the study design and contributed to the writing of the paper.

# A Case Study on a Specification Approach using Activity Diagrams in Requirements Documents

Martin Beckmann  
Technische Universität Berlin, Germany  
martin.beckmann@tu-berlin.de

Andreas Vogelsang  
Technische Universität Berlin, Germany  
andreas.vogelsang@tu-berlin.de

Christian Reuter  
Daimler AG, Germany  
christian.c.reuter@daimler.com

**Abstract**—Rising complexity of systems has long been a major challenge in requirements engineering. This manifests in more extensive and harder to understand requirements documents. At the Daimler AG, an approach is applied that combines the use of activity diagrams with natural language specifications to specify system functions. The approach starts with an activity diagram that is created to get an early overview. The contained information is then transferred to a textual requirements document, where details are added and the behavior is refined. While the approach aims to reduce efforts needed to understand a system's behavior, the application of the approach itself causes new challenges on its own. By examining existing specifications at Daimler, we identified nine categories of inconsistencies and deviations between activity diagrams and their textual representations. In a case study, we examined one system in detail to assess how often these occur. In a follow-up survey, we presented instances of the categories to different stakeholders of the system and let them assess the categories regarding their severity. Our analysis indicates that a coexistence of textual and graphical representations of models without proper tool support results in inconsistencies and deviations that may cause severe maintenance costs or even provoke faults in subsequent development steps.

## I. INTRODUCTION

Complex software systems, which e.g. can be found in distributed embedded systems in automotive electronics, require model-based and system-oriented development approaches [1]. Using graphical models for specification manages complexity and improves reusability and analytical capabilities [2], [3]. Although graphical models provide a suitable means to specify and understand dependencies and procedural behavior of a system, in industry they are usually accompanied by a textual representation. Previous work has shown the need for a continuous systems engineering environment, where referring or constitutive documents are essential to work on complex software systems [4]. Also the combined use of graphical diagrams and textual descriptions is considered beneficial for the requirements management process [5], [6]. In addition, for industrial applications, tool support and model exchange for graphical models is still not standardised and, as a result, manufacturer/supplier handover is still performed by textual documents. This is especially important, since these textual documents often serve as the basis for legal considerations between the contractors [6], [7]. Also, due to different backgrounds of the stakeholders, not everyone is capable of understanding the graphical models [8]. Thus, the

information contained in a model needs to be written in words to be appropriately reviewable [9].

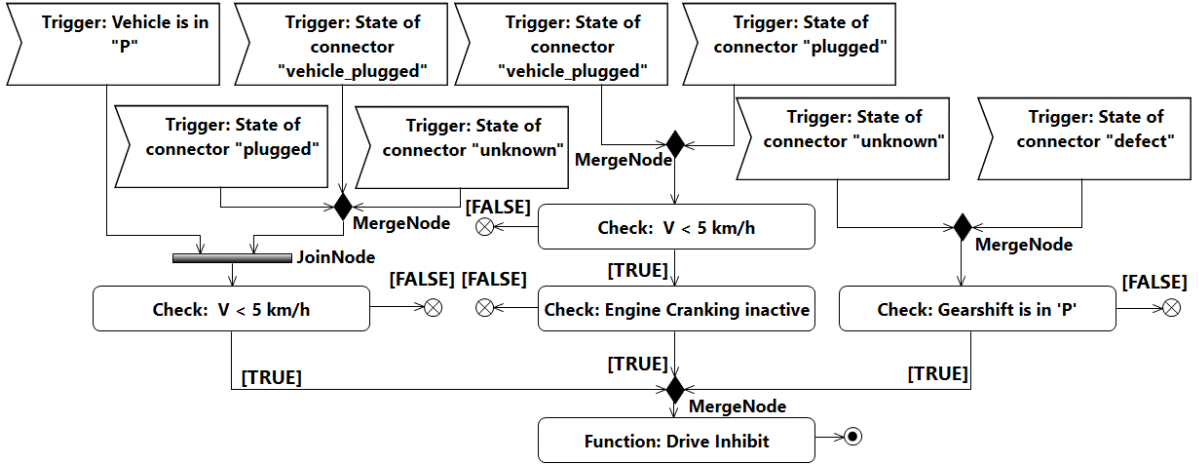
Daimler applies an approach, where, as a first step, a UML activity diagram [10] is created for each function to describe the function's activation and deactivation by triggers and conditions. This kind of description is also known in literature to formulate textual natural language requirements [11]. Textual representations of the activity diagrams along with the diagrams themselves are then transferred into a requirements document for everyone to understand and for ongoing development. The transfer of the model into the requirements document is done manually. This is an error-prone task. Besides, the ongoing development using the requirements document might also cause inconsistencies between the activity diagrams and the document, in case the activity diagrams are not kept up-to-date.

We are interested in understanding what types of inconsistencies and quality issues are introduced by using activities and a textual specification alongside one another and how severe these issues are. If the approach itself introduces more severe issues than expected benefits, this is a strong argument for automated consistency checks and quality assurance.

For this purpose, we examined 36 vehicle functions of one system at Daimler that was specified by the introduced approach. As a result, a number of inconsistencies between the requirements document and the activity diagram were found.

All of these findings resulted in nine different categories of quality issues. We found occurrences of these categories in all of the examined functions. The categories are introduced in detail as well as the amount of findings in the examined system. Also we presented the quality issues to different stakeholders of the system, who assessed their severity. The occurrences, that are perceived as major quality issues, are present in 78% of the vehicle functions.

The paper's structure is given in the following manner. The next section details the approach, that is used to specify the system's functions. The third section introduces the nine categories, that were found examining the activity diagrams and their respective textual representation in a requirements document. In the fourth section the study design is explained. Section five presents the results of the study and the conducted survey. The sixth section discusses the results and possible means to avoid the discovered quality issues. The last section concludes this work.

(a) Activity diagram of the Function *Drive Inhibit*

ID	Text	Level	Type
1000	<b>1.1.1.1.1.1 Drive Inhibit</b>	6	Function
1236	State of connector "unknown" OR State of connector "defect" OR	7	Trigger
1237	Vehicle Gear selector is in position "P" AND	8	Check
1113	Engine Cranking inactive OR	8	Check
1111	State of connector "plugged on vehicle side" ("VEH_PLUGGED") OR "plugged on vehicle and EVSE side" ("PLUGGED"). OR	7	Trigger
1112	Vehicle velocity is below 5 km/h	8	Check
1114	Vehicle Gear selector is in position "P" OR	7	Trigger
1232	Vehicle velocity is below 5 km/h	8	Check
1233	State of connector "plugged on vehicle side" OR State of connector "plugged on vehicle and EVSE side" OR State of connector "unknown" AND	7	Trigger
1238	Vehicle velocity is below 5 km/h	8	Check

(b) Textual specification of the Function *Drive Inhibit*

Fig. 1: Activity diagram and the specification text of a function

## II. BACKGROUND

The Daimler approach used to specify functions of a system employs UML activity diagrams. These activity diagrams are the first step of specifying a new function. They are used to get an early overview of the desired function behavior with a special focus on the functions activation, execution conditions, functional paths, and deactivation. The information contained in the activity diagram as well as the activity diagram itself is then transferred to a textual requirements document. This transfer is necessary, since this textual requirements document is the central artefact for further development. Besides, the textual document contains additional and more detailed information as well as statements about its context, which relates this approach to Literate Modelling [8].

Fig. 1 shows an exemplar specification as we have found it at Daimler. The example consists of an activity diagram and

its textual representation in the requirements document. In the following, we will explain the example and also the contained quality issues. In the remainder of this work, an element refers to an entity contained in an activity diagram, whereas an object in the text refers to an entity contained in the requirements document.

Fig. 1a displays the activity diagram of the function *Drive Inhibit*. The actual behavior of the activated function is described in the *Action* node labeled with *Drive Inhibit* (bottom of the diagram). The function's activation is described by a combination of triggers and checks for conditions. For triggers, the *AcceptEventAction* element is used. The checks are modeled as *Action* elements. If the condition of a check is not fulfilled, the flow ends (*FlowFinal*). The triggers and checks are connected by *ControlNodes* such as *JoinNodes* and *MergeNodes*. *JoinNodes* act as synchronisation points and can



be interpreted as AND operators in terms of propositional logic. *MergeNodes* represent OR operators. Once the actual functionality of the function is executed, *ActivityFinal* elements designate the end of an activity.

The corresponding chapter in the textual requirements document is displayed in Fig. 1b. Each row in the document represents an object, which is described by a set of attributes (columns). The *ID* attribute contains a unique identifier of the object. The *Text* attribute is a textual description of the object and is supposed to be equal to the text of the corresponding element in the activity diagram. The *Level* is an attribute to structure the document hierarchically. It is derived from the structure of the activity diagram. The *Type* attribute of each text object is supposed to be equal to the type of its corresponding element in the diagram. These attributes are needed to display the relevant information of the activity diagram in the requirements document. Besides the given attributes, the document contains additional attributes used for further development.

There are many possibilities to display different aspects of an activity diagram as text. An exact textual representation as presented in [12] is not desirable, since it lacks proper readability and comprehensibility for those, unfamiliar with activity diagrams. Instead, the used textual representation focuses on the propositional logic, readability, and the recognition value of the structure of the activity diagram. This is implemented by copying the text of the elements of the diagram into distinct objects. Propositional logic operators such as OR and AND are used as strings in the *Text* attributes of the objects to realise the logic statements of the activity diagram. The operators at the end of an object's text connect the object with the following object on the same level of the document hierarchy. For instance, in Fig. 1b, the object with *ID 1236* is connected via an OR with the object with *ID 1111* because it is the next object on the same hierarchical level. Besides the propositional logic purposes, the different levels of the documents are used to display the belonging of the elements within the activity diagram. For example, the check *Vehicle Gear selector is in position "P"* (*ID 1237*) is executed after one of the triggers contained in the object with the *ID 1236* occurred. Hence, it appears one level below. This is important, since there might be more than one check associated with a set of triggers, as can be seen in the object in the text with *ID 1113*.

The transfer of information from the activity diagram to the requirements document is a manual process. This might lead to inconsistencies between the activity diagram and the requirements document and other quality issues as can be seen in Fig. 1. Amongst others, these inconsistencies and quality issues are presented in the next chapter.

### III. IDENTIFIED QUALITY ISSUES

A preliminary examination of a set of requirements documents at Daimler revealed a number of quality issues. The quality issues are inspired by standards such as ISO/IEC/IEEE 29148 [13], CMMI-Dev [14] and Automotive SPICE [15]. We grouped these quality issues into nine different categories. The

TABLE I: Categories of identified quality issues

Category name	Description
Missing Tracing	There is no information to trace an element to its corresponding object in the text or vice versa.
Missing Element/Object	Either the activity diagram or the requirements document contains entities, which the other does not contain.
Incorrect Logic	The propositional logic of the activity diagram deviates from the requirements document or the logic connections are not clear.
Textual Differences	Elements and their corresponding objects in the text exhibit textual differences.
Redundant Element	The activity diagram contains multiple elements, which have the same type and the same text.
Non Atomic Element/Object	Either an element or an object contains multiple statements. This might appear in both the requirements document or the activity diagram.
Wrong Placement	The placement of an element in the activity diagram does not match with the placement of the corresponding object in the requirements document.
Unnecessary Repetition	There are multiple objects in the requirements document, which are derived from one single element.
Wrong Type	The type of the element does not match the type of the corresponding object.

categories and their descriptions are listed in Table I. The categories cover the relation between the activity diagrams and the requirements document. Some of them only appear either in the diagram or the text, but still have an influence on the respective other artefact. We will explain some of the categories by examining the example in Fig. 1.

The category **Incorrect Logic** is present in the objects in the document with the *ID 1113* and *ID 1233*. Both objects end with an operator, for which it is not clear which object they refer to. Neither of them has a successor on the same level below their respective parent object. The object with the *ID 1236* is the parent object of two objects (*ID 1237*, *1113*) containing checks.

**Textual Differences** can be found (amongst others) between the triggers in the objects with the *ID 1111*, *1233* and their corresponding elements of the diagram.

There are multiple **Redundant Elements** in the diagram such as the checks  $V < 5 \text{ km/h}$  and the triggers *State of connector "plugged"*. In this example, the appearance of redundant elements in the diagram can be avoided by inserting additional *ControlNodes* and restructuring the activity diagram [16].

While all elements of the diagram are atomic, the requirements document contains several **Non Atomic Objects** (*ID 1236*, *1111*, *1233*). These objects incorporate multiple assertions that are connected by propositional logical operators. This is both an issue in the requirements document as well as a

deviation between the activity diagram and the requirements document.

The elements in the diagram, corresponding to the object with the *ID 1236*, are followed by the diagram element *Check: Gearshift is in 'P'*. In the document the corresponding object (*ID 1236*) is the parent object of an additional check (*ID 1113*). The additional check in the document is elsewhere in the activity diagram. This situation is denoted as the category **Wrong Placement**.

The requirements document contains *Check:  $V < 5$  km/h* three times (*ID 1112, 1232, 1238*). But there are only two elements in the activity diagram. Hence, two of the objects in the document refer to one single element in the diagram. This is an instance of the **Unnecessary Repetition** category and can be avoided by grouping the objects accordingly.

We used these categories to find out how many quality issues occur in the vehicle functions of a system at Daimler and how much these occurrences influence the quality of the requirements document.

#### IV. STUDY DESIGN

To find out how often instances of the identified categories appear in a system and to understand how this system is impacted by these occurrences, we conducted a case study. We designed the study along the recommendations of Runeson and Höst [17]. Our research objective is:

**Research Objective:** We want to find out which problems the coexistence of textual and graphical representations of models implicate and how severe these problems are.

To reach this aim, we pursue four research questions (RQ):

**RQ1: How many occurrences of the categories can be found in a system?** To assess the influence of the occurrences of the categories, we need to know how many instances of each category occur. The number of occurrences of each category is one of the major contributing factors for the impact on the quality.

**RQ2: Are the stakeholders of the system aware of these occurrences?** We want to find out whether the stakeholders know about occurrences of existing quality issues. This gives us an idea on whether these occurrences have already been noticed. This is a first indication on how severe the occurrences are perceived.

**RQ3: Do the stakeholders agree that these occurrences are quality issues?** After we found out whether the stakeholders are aware of deviations and inconsistencies between activity diagrams and their textual representation, we want to know whether they agree with our assessment that a certain situation is in fact a quality issue. This is of interest since different backgrounds and responsibilities of the stakeholders might result in different opinions on what quality issues are.

**RQ4: How do the stakeholders assess the severity of these occurrences?** Besides the number of occurrences, the severity of an occurrence is the second major contributing factor of its impact on quality. Hence, the answer to this question is needed to evaluate the severity of each category.

**Study Object:** We examined a specific subsystem of a car developed at Daimler. The examined subsystem is responsible for charging the high-voltage batteries of Plug-in Hybrid Electric Vehicles and Battery Electric Vehicles. As such the system contains requirements that are relevant for safety as well as for usability. The system's requirements are documented in specification artefacts (activity diagrams and their textual representations) resulting from the approach described in Section II. The requirements document of the system contains a total number of 46 functions. In our study, we only considered 36 of these functions, since some functions were not specified using the approach and hence did not contain activity diagrams. Other functions were discarded because the corresponding text did not adhere to the pattern for the textual representation.

**Data Collection:** To answer RQ1, we manually searched for instances of the quality issue categories in all activity diagrams and their respective textual representation. To increase reliability and to avoid that occurrences are overseen, two researchers conducted this examination independently. The results were then compared and missing occurrences were complemented.

To answer RQ2 – RQ4, we conducted a survey amongst the stakeholders of the system that relates to the results of our manual document inspection. A total of seven stakeholders participated in the survey. Of the seven participants, three are authors of requirements documents for specific system components, two are responsible for testing, and one is an author of the requirements document of the system's functions. The last remaining participant is involved in developing the methodology that is used for the specification process.

As part of the survey, we presented two occurrences of each quality issue category as samples to the participants. The samples originated from specifications of several vehicle functions of the system. We selected actual samples of the system rather than abstract examples to improve the comprehensibility of each category and to give a better impression on the actual effect of the involved activity diagram and its corresponding textual representation. Each sample contains both of them. The issues in the activity diagram and the textual representation are highlighted by using colored frames. Besides, each sample is accompanied by a text explaining why the presented situation might have a negative effect on the quality. However, the concrete name of the category is not shown. This prevents the stakeholders from assessing the category rather than the concrete example. The rationale is to find out, whether the severity of different instances of one category is perceived differently. Most of the examined vehicle functions contain instances of multiple categories. Hence, some of the presented samples show the same vehicle function highlighting a different category each time. There was no specific order in which the samples were presented. However, the two samples of each category were never presented consecutively. The reason for this is to mitigate the influence of previous decisions.

The stakeholders were asked to answer the following survey questions (SQ) for each sample:

**SQ1: Were you aware of the existence of this finding?**

We first needed to know, whether the stakeholder had already recognised the presented situation of the sample. The participants could answer this by selecting *yes* or *no*. This aims at answering RQ2.

**SQ2: Do you think this sample is in fact a quality issue?**

This question is used to find out whether the stakeholder actually recognises the presented situation as a quality issue, now that it has been presented as such. The participants could answer this by selecting *yes* or *no*. The question aims at answering RQ3.

**SQ3: When would you fix this quality issue?**

This was asked to assess the severity of the quality issue. We presented four options to answer this question to the stakeholders: *immediately* (during the same project iteration), *soon* (next time the function is edited), in the *long term* (when there is time to clean up the document), or *never*. The question aims at answering RQ4.

## V. STUDY RESULTS

### A. RQ1: Occurrences of Quality Issues

Table II shows the total number of occurrences we found for each category. The third column shows in how many functions we found quality issues of each category and the last column shows the average number of findings per function. The results show that we found at least 10 occurrences of each category in the 36 examined functions. Moreover, the Missing Tracing occurred in all elements of all functions, which means that we found no trace links to diagram elements at all. Secondly, we found missing elements or objects in the text or the diagram in 78% of the functions. In total, 126 elements and objects were missing, which accounts for 3.5 missing elements and objects per function on average. We found Incorrect Logic and Textual Differences in more than half of the examined functions. Textual Differences accounted for 43 findings in total. Wrong Type, Unnecessary Repetition, and Wrong Placement were the categories that appeared the least, although we still found them in about a quarter of all functions.

**Discussion:** The reported numbers show that a manual transition process between graphical activity diagrams and textual requirements documents bears a high risk of introducing deviations and inconsistencies, which we characterised as quality issues. Our analysis shows that this process is especially prone to missing out elements or objects, introducing incorrect logic, and textual differences. Whether these quality issues are really perceived as such by the stakeholders is examined in RQ2–RQ4.

### B. RQ2: Awareness of Quality Issues

The distribution of answers to SQ1 is displayed in Fig. 2. There are two bars for each category. Each bar represents the answers for one sample. In general, the presented samples were mostly unknown to the participants. There are five samples, where all participants mentioned that they were unaware of

TABLE II: Occurrences of quality issues per category ordered by frequency of occurrences in functions.

Category name	Findings	Number and ratio of functions with findings	Average number of findings per function
Missing Tracing	all <sup>1</sup>	36 (100 %)	-
Missing Element/Object	126	28 (78 %)	3.5
Incorrect Logic	29	22 (61 %)	0.8
Textual Differences	43	20 (56 %)	1.2
Redundant Element	24	15 (42 %)	0.66
Non Atomic Element/Object	18	14 (39 %)	0.5
Wrong Placement	18	10 (28 %)	0.5
Unnecessary Repetition	15	9 (25 %)	0.42
Wrong Type	10	8 (22 %)	0.28

their existence. For 12 samples, six out of seven participants stated that they were not aware of the existence. One sample belonging to the category Non Atomic Element/Object was known by two of the participants. It is worth noting that every time a situation was answered with *yes* at least once, a certain participant was always amongst those. This participant is the one involved in the development of the methodology.

**Discussion:** The answers to this question show that the stakeholders are mostly unaware of the presented occurrences. This fact explains, why we found these issues in the first place. Nevertheless we were quite surprised by these results. One possible explanation is that the selected participants were not involved in the development of functions from which we selected the samples. Since we selected the samples from a number of functions and a participant usually contributes to more than one function, this explanation is not very likely. An alternative explanation is that the selected samples belong to vehicle functions that are not frequently examined. Hence, their existence might have not been noticed. We had no information about the frequency of changes for functions. Another possibility is that the presented situations are not perceived as quality issues. Whether the samples are not perceived as quality issues so far or not at all is the subject of RQ3.

### C. RQ3: Agreement on Quality Issues

The answers to SQ2 are displayed in Fig. 3. The diagram is composed the same way as the diagram in Fig. 2. 14 out of the 18 samples were assessed to be quality issues by the majority of the participants. For three samples, all participants decided that these samples actually are quality issues. This applies to both samples of the category Wrong Placement. The other sample that all participants classified as a quality issue belongs to

<sup>1</sup>In the examined specifications, no tracing links between diagram elements and textual objects were defined.

Were you aware of the existence of this finding?

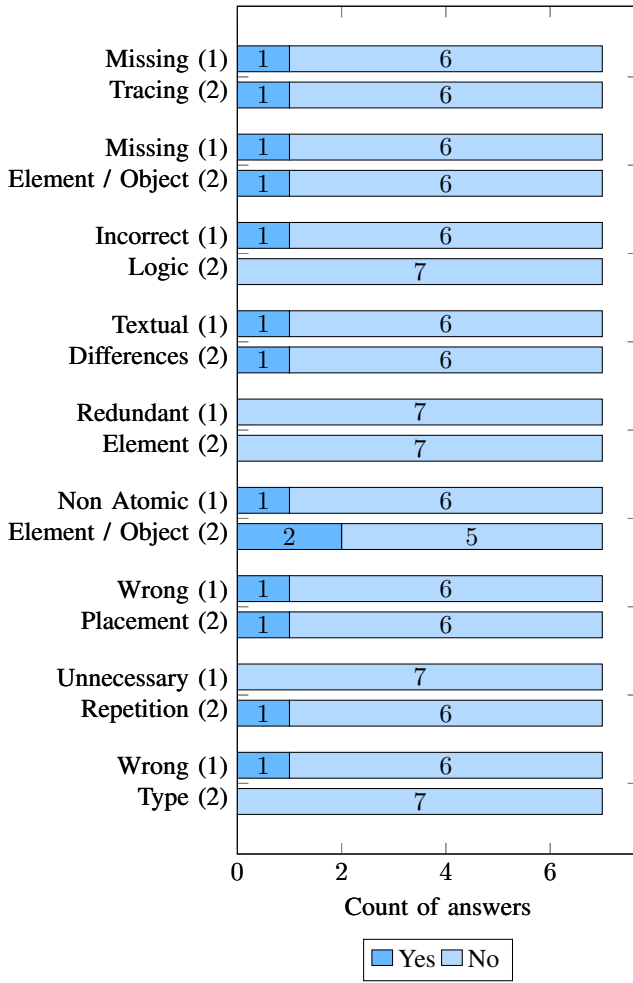


Fig. 2: Answers to SQ1

Do you think this sample is in fact a quality issue?

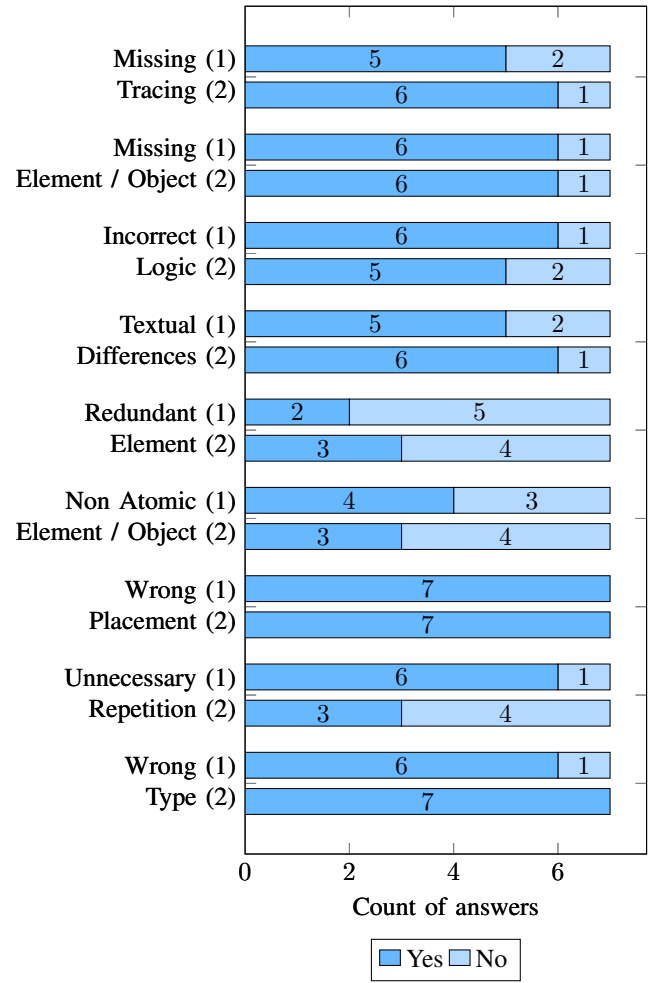


Fig. 3: Answers to SQ2

the category Wrong Type. Those two categories in addition to Missing Element/Object show the highest agreement amongst the participants to actually be quality issues. The samples with the lowest number of participants seeing them as quality issues are in the categories Redundant Element, Non Atomic Element/Object, and Unnecessary Repetition. Whereas most samples of one category were assessed similarly, the samples of category Unnecessary Repetition showed a large deviation. Its first sample is amongst those with the highest approval (six yes to one no), while the other is amongst the lowest (three yes to four no). We have no explanation for this result, since the two samples are very similar. Hence, further investigation is needed to assess, whether the stakeholders did not fully understand the presented situation or whether some of the stakeholders had a specific reason to assess the second sample differently.

**Discussion:** The answers to this question show that there is a high level of agreement that the samples of the categories Missing Element/Object, Wrong Placement, and Wrong Type in fact constitute quality issues. For the categories Redundant Element and Non Atomic Element/Object, many participants

assessed the identified samples as not being quality issues. This shows that the participants may have a different understanding of quality in these cases. Overall, there are only small differences between the samples within one category. This indicates that the perception might be the same for all other occurrences as well. How stakeholders assess the severity and whether the severity of the samples of one category are also similar is the subject of RQ4.

#### D. RQ4: Severity of Quality Issues

The answers to SQ3 are displayed in Fig. 4. As in the diagrams in Fig. 2 and Fig. 3 the answers for both samples of each category are displayed. The category, where the samples were perceived as most severe is Wrong Placement. At least five participants answered that they would fix these situations *immediately*. The remaining participants mentioned that they would fix them *soon*. For the categories Missing Element/Object, Textual Differences, and Wrong Placement no one answered with the option *never*. This means that all participants identified a need for improvement, which is in line with the

result of SQ2 where most participants assessed the samples of these categories as quality issues. The sample with the lowest severity is in the category Redundant Element (one *immediately*, two *soon*, one *long term*, three *never*). This is also in line with the results of SQ2. Some samples were assessed quite diverse. For example, the first sample of the category Non Atomic Element/Object would be fixed *immediately* by four participants, while two participants would *never* fix them. This sample consists of three propositional logic statements, that are all connected via an OR. In the requirements document all of the statements are contained in one single entry, while in the activity diagram, there are three distinct elements connected by a *MergeNode*. For the category Incorrect Logic, the severity of the two samples were assessed quite differently. While most participants agreed that they would fix the first sample *immediately*, two participants stated that they would *never* fix the second sample.

**Discussion:** The fact that, aside from one sample, the majority of the participants answered at least with *soon*, suggests that the identified categories are not only quality issues, but have to be considered for future development of the requirements document. However, some samples were rated with the option *never*. Especially for the samples of the categories Incorrect Logic it is interesting that some participants answered to *never* or only in the *long term* fix a quality issue, even for samples that reflect obvious deviations between the diagrams and the corresponding text (i.e., diagram and text describe different behavior). A possible explanation for this might be that these participants have a different understanding of the diagram's and text's semantics compared with us or that they just *use* them differently (e.g., they do not use the text to understand the function's behavior but only to look up some details). More than half of the participants answered to both samples of the categories Missing Element/Object and Wrong Placement with *immediately*. On top, no one answered with the option *never*. Therefore, we consider occurrences in the categories Missing Element/Object and Wrong Placement as major quality issues. Hence, 78% of the functions contain at least one major quality issue. This is also ratio of occurrences of the category Missing Element/Object. There was no occurrence of Wrong Placement without an occurrence of the category Missing Element/Object in the same function.

## VI. DISCUSSION

We conclude from our results that in this case a specification approach based on coexisting graphical and textual representation of specification models bears a high risk of introducing quality issues. More specifically, in our investigation of quality issues between activity diagrams and their textual representation, we assessed that Missing Tracing, Missing Element/Object, Textual Differences and Incorrect Logic are the most frequent quality issues.

Although most stakeholders were unaware of the occurrences that we presented to them, they agreed that those occurrences are in fact negatively impacting the quality of the requirements document. Since the samples of each category were always

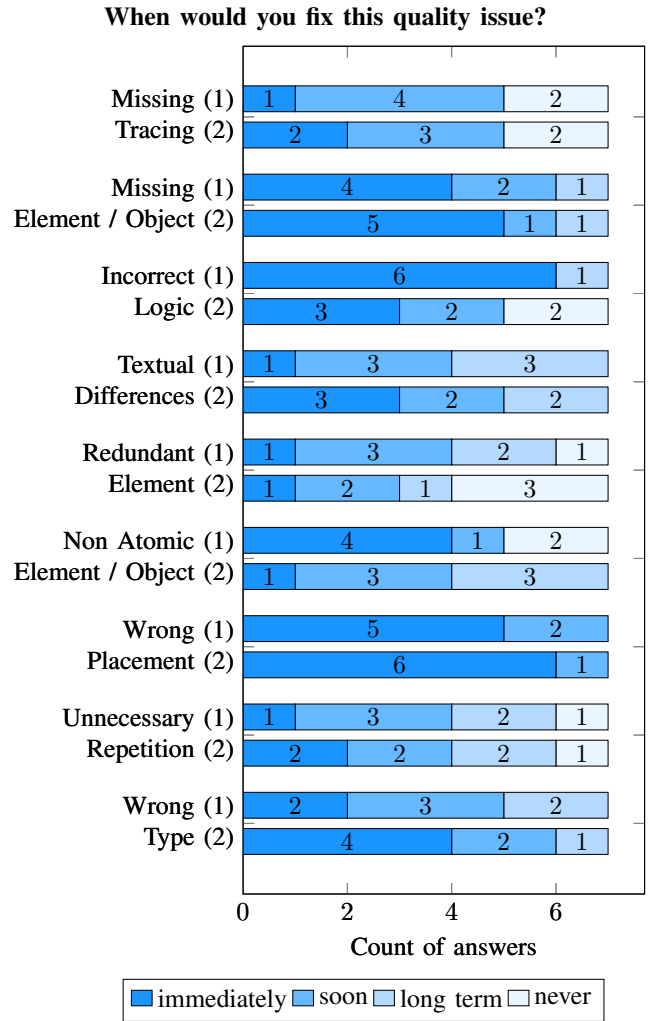


Fig. 4: Answers to SQ3

rated similarly by the participants, we may generalise this assessment to an assessment of the category itself. By this, we conclude that our defined categories Wrong Placement, Missing Element/Object, and Wrong Type definitively constitute quality issues. For the categories Redundant Element, Non Atomic Element/Object, and Unnecessary Repetition, the participant's opinions diverged. This challenges our initial hypothesis that findings of these categories are indeed quality issues.

The findings of RQ3 are consistent with the findings of RQ4. Categories Redundant Element and Non Atomic Element/Object are perceived as the least severe quality issues. In retrospect, this could also explain why participants were not aware of these quality issues since they did not perceive them as such so far. Also, the findings consistently suggest, that the categories Missing Element/Object and Wrong Placement are the most severe. This is especially important, since Missing Element/Object occurs the most often after Missing Tracing. Since Missing Tracing is a category appearing in every object of the document, it is hard to understand why it was rated as

a quality issue by so many. The findings of RQ4 also indicate, that there is a need for complete tracing between the artefacts of the two representations. The reason for the missing tracing is a consequence of the manual transition from the modeling tool to the RE management tool. This leads to the question why this approach was implemented without proper tool support in the first place. Possible solutions to this problem and the quality issues in general are presented in Section VI-A.

Initially, we also expected different perceptions of quality issues between participants of the different stakeholder groups. However, this was not confirmed in the study. The results were almost identical for participants with different stakeholder roles. There was only one exception. The expert on methodology was the only participant that was aware of most of the quality issues. At the same time that stakeholder only disagreed with us on three different samples originating from three different categories. Also that person only answered with *immediately* three times and is the person, who answered with *never* most often (four times).

Although it was not the scope of our research, comparing the timestamps of the activity diagrams with the dates of the baselines of the requirements document suggests that the activity diagrams are primarily used at the beginning. This might be caused by the specification process and is in accordance with other research findings [18].

#### A. Possible solutions

A possibility to reduce the number of quality issues, that arise by the manual transferal, might be the application of reviews as reported by Terzakis [19]. In order to avoid quality issues resulting from deviations between graphical and textual representations altogether, automatic approaches can be used to keep the diagram and the requirements document in sync. The generation of requirements documents from graphical models is an established approach [20]. Different approaches were suggested for specific graphical models. For instance, Maiden et al. [7] use i\* models to derive requirements and De Landsheer et al. [21] propose a similar approach for the KAOS goal-oriented method. Fockel and Holtmann [22] present a model-driven RE approach with tool support that provides synchronisation capabilities for the applied RE models and their textual representations. Still, these approaches are specialised to specific techniques during requirements elicitation and management.

Other than that, there are also approaches, that derive textual requirements or a structure for parts of requirements documents from different types of UML/SysML diagrams. Robinson-Mallett [23] shows how Statecharts and Block Diagrams can be used to create a structure for a requirements document. Berenbach [24] introduces an algorithm that derives a structure for a requirements document from use case diagrams. In an additional work [25], the possibility of synchronisation is mentioned, although it is limited to textual changes. In addition, the approach is restricted to diagrams that adhere to certain guidelines. Since all these approaches do not use activity diagrams, they are not applicable to the presented specification

approach. The approach presented by Drusinsky [26] supports activity diagrams, however, only for UML-1. Additionally, only the generation of actual requirements is addressed but not the creation of a requirements document structure.

Both the improvement of the manual transferal as well as automated approaches might benefit from an adjusted representation of the activities in the requirements document. A more sophisticated structure might mitigate some of the quality issues, while maintaining proper readability.

Another possibility is the use of Projectional Editors, which automatically edit different projections of a common underlying model, in this case the activity diagram and its textual representation. However, this possibility may require substantial efforts and experienced developers [27]. Hence, a custom-made and lightweight synchronisation solution might be well suited to prevent the mentioned quality issues for the used specification approach.

A more rigorous solution to the problem of inconsistent textual and graphical representations is to understand the development process as a stepwise refinement of natural language requirements to models that detail and formalise the original requirements [28]. In these processes, changes must be followed by a pipeline of updates along the chain of refinement models.

#### B. Threats to validity

The identification of quality issue candidates was performed manually. Manual processes are error-prone and also subjective to some degree. In order to mitigate this threat, two authors analysed the documents independently. To reduce the subjectivity, we developed a precise description of the quality issue candidates. We did not compute an inter-rater agreement, however, after the independent classification, we only had to discuss six occurrences for which the classification was different. Also, we cannot claim, that the quality issues we found cover all aspects of deviations between an activity and its textual representation.

We did not have any information about the development of the models and the documents over time. Including these information might lead to different classifications in some cases. For instance, certain findings that we classified as Missing Element/Object might actually be elements that were strongly altered over time so that we were not able to identify the relation between the elements any more. In this case, the finding would actually fall into the category Textual Differences.

Besides, the analysis was done without any explicit domain knowledge of the used system. This might have led to misinterpretations regarding the categories. This also affects the categories Missing Element/Object and Textual Differences as we might not have recognised mere textual changes as such. Instead these occurrences ended up in the category Missing Element/Object.

Due to the large number of findings, we only presented two representative findings of each category to the stakeholders. We used the assessment of these findings as proxy for an evaluation of the whole category. Our results show that the two samples

of each category, in general, were assessed similarly. Still, it is possible that the selected samples were perceived as more or less severe than other samples of the same category would have been. The participating stakeholders were selected by the third author who is also actively participating in the development of the examined system. We did not follow specific selection criteria, except that participants must work actively on the examined systems. However, the group of study participants only represent a subset of all people working actively with the requirements documents. Furthermore, not all of those we contacted, reported back to us in time. We originally contacted twelve participants of which seven answered our survey.

Our results indicate that quality issues arise from the presented specification approach that is used in some projects at Daimler. To answer our research questions, we only had access to one system developed with this approach. Hence, the generalisability of our findings are limited. Discussing the results with the stakeholders at least left the impression that the results are not surprising to them and that they would expect similar findings also in other systems developed with this approach.

## VII. CONCLUSION AND FUTURE WORK

In this paper, we presented possible quality issues, that may arise when using a certain specification approach, that we encountered at Daimler. The approach incorporates UML activity diagrams in requirements documents. Those activity diagrams are accompanied by a textual representation of the diagrams. The textual representation is edited and further refined during ongoing development.

We conducted a case study on a real system. The purpose of this study was twofold. First we assessed the total number of occurrences of possible quality issues in the requirements document of the system. The second part is a survey amongst the stakeholders. The aim was to find out, whether they agree, that the quality issues we identified are in fact quality issues and how they rate the severity of preselected samples.

All of the examined functions were affected, since there was no tracing present between the activity diagram elements and the objects of the text. Other than that we found between 10 and 126 occurrences of each identified quality issue. The survey showed, that the stakeholders were unaware of the existing quality issues. Nevertheless, the majority of them agreed, that seven out of nine identified quality issues are in fact issues impacting the quality of the requirements document.

For all but one sample, the majority of the stakeholders saw the need to fix the quality issues at latest during the next time the function is edited. However, there are eight samples, where at least one stakeholder saw no need in fixing the issue.

Since there was only one system available, the generalisability is limited. The findings require additional validation by repeating the case study with a different system and more participants.

An aspect, that was out of scope of this work, is the influence of the identified quality issues on following development stages. The case study assessed the number of occurrences in a certain

requirements document and the severity of the quality issues, but not the resultant consequences. Hence, it needs to be addressed how these quality issues effect the development of the final product and future products, that reuse the existing requirements document.

## REFERENCES

- [1] M. Broy, "Challenges in automotive software engineering," in *Proceedings of the 28th international conference on Software engineering*. New York, NY, USA: ACM, 2006.
- [2] L. Apfelbaum and J. Doyle, "Model Based Testing," in *Software Quality Week Conference*, 1997.
- [3] A. Vogelsang, S. Eder, G. Hackenberg, M. Junker, and S. Teuffl, "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'14)*, 2014.
- [4] C. Reuter, "Variant Management as a Cross-Sectional Approach for a Continuous Systems Engineering Environment," in *Proceedings of the 8th Grazer Symposium Virtual Vehicle*, 2015.
- [5] A. M. Davis, *Just Enough Requirements Management: Where Software Development Meets Marketing*. New York, NY, USA: Dorset House Publishing Co., Inc., 2005.
- [6] E. Sikora, B. Tenbergen, and K. Pohl, "Industry needs and research directions in requirements engineering for embedded systems," *Requirements Engineering*, vol. 17, no. 1, 2012.
- [7] N. A. Maiden, S. Manning, S. Jones, and J. Greenwood, "Generating requirements from systems models using patterns: a case study," *Requirements Engineering*, vol. 10, no. 4, 2005.
- [8] J. Arlow, W. Emmerich, and J. Quinn, "Literate Modelling — Capturing Business Knowledge with the UML," in *International Conference on the Unified Modeling Language*. Springer, 1998.
- [9] R. F. Goldsmith, *Discovering Real Business Requirements for Software Project Success*. Artech House, 2004.
- [10] Object Management Group (OMG), "OMG Unified Modeling Language (OMG UML), Version 2.5," OMG Document Number formal/2015-03-01 (<http://www.omg.org/spec/UML/2.5/>), 2015.
- [11] D. Firesmith, "Generating complete, unambiguous, and verifiable requirements from stories, scenarios, and use cases," *Journal of Object Technology*, vol. 3, no. 10, 2004.
- [12] D. Flater, P. Martin, and M. Crane, "Rendering UML Activity Diagrams as Human-Readable Text," Tech. Rep., 2009.
- [13] The Institute of Electrical and Electronics Engineers, Inc., "ISO/IEC/IEEE 29148:2011, Systems and Software Engineering – Life cycle processes – Requirements Engineering," 2011.
- [14] CMMI Product Team, "CMMI for Development, Version 1.3," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep., 2010. [Online]. Available: <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9661>
- [15] VDA QMC Working Group 13 / Automotive SIG, "Automotive SPICE Process Assessment / Reference Model," 2015.
- [16] M. Beckmann and A. Schlutter, "Automatische Duplikateliminierung in Aktivitätsdiagrammen von Fahrzeugfunktionen," in *INFORMATIK 2016, Lecture Notes in Informatics (LNI)*, 2016.
- [17] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, 2009.
- [18] R. Hebig, T. Ho-Quang, G. Robles, M. Fernandez, and M. Chaudron, "The Quest for Open Source Projects that Use UML," in *19th International Conference on Model Driven Engineering Languages and Systems*, 2016.
- [19] J. Terzakis, "The Impact of Requirements on Software Quality across Three Product Generations," in *21st IEEE International Requirements Engineering Conference (RE)*. IEEE, 2013.
- [20] J. Nicolás and A. Toval, "On the generation of requirements specifications from software engineering models: A systematic literature review," *Information and Software Technology*, vol. 51, no. 9, 2009.
- [21] R. De Landtsheer, E. Letier, and A. Van Lamsweerde, "Deriving tabular event-based specifications from goal-oriented requirements models," *Requirements Engineering*, vol. 9, no. 2, 2004.

- [22] M. Fockel and J. Holtmann, "A requirements engineering methodology combining models and controlled natural language," in *4th International Model-Driven Requirements Engineering Workshop (MoDRE)*. IEEE, 2014.
- [23] C. L. Robinson-Mallett, "An approach on integrating models and textual specifications," in *2nd International Model-Driven Requirements Engineering Workshop (MoDRE)*. IEEE, 2012.
- [24] B. Berenbach, "The Automated Extraction of Requirements from UML Models," in *11th IEEE International Requirements Engineering Conference (RE)*. IEEE, 2003.
- [25] B. Berenbach, "Comparison of UML and Text based Requirements Engineering," in *Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, ser. OOPSLA '04. New York, NY, USA: ACM, 2004.
- [26] D. Drusinsky, "From UML activity diagrams to specification requirements," in *IEEE International Conference on System of Systems Engineering (SoSE)*, 2008.
- [27] T. Berger, M. Völter, H. P. Jensen, T. Dangprasert, and J. Siegmund, "Efficiency of projectional editing: A controlled experiment," in *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, 2016.
- [28] W. Böhm, M. Junker, A. Vogelsang, S. Teufl, R. Pinger, and K. Rahn, "A formal systems engineering approach in practice: An experience report," in *1st International Workshop on Software Engineering Research and Industrial Practices (SER&IPs'14)*, 2014.



### Part III

## AUTOMATIC GENERATION OF TEXTUAL REPRESENTATIONS FROM ACTIVITY DIAGRAMS



## REMOVAL OF REDUNDANT ELEMENTS WITHIN UML ACTIVITY DIAGRAMS

---

Published in 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS) (p. 334 - 343) [22].

### TERMS OF USE

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

### BROADER CONTEXT WITHIN THE THESIS

In Chapter 3 it became apparent that inconsistencies between the representations and other quality issues affect the work of users. Chapter 4 confirmed these findings, as the amount and severity of appearing inconsistencies and other quality issues presents a major difficulty for the applicability of the approach in practice. The results in Chapter 3 and Chapter 4 suggest that the textual representation should be automatically generated from the activity diagrams. Through this method the occurrence of inconsistencies and other quality issues can be avoided. At the same time, such a *model-to-requirements* approach (see Subsection 2.4.1) also ensures the existence of trace links between representations.

To allow for automation, activity diagrams without redundant elements (in particular *ExecutableNodes*) are required as input. This is necessary, as such redundant elements impede the unambiguous identification of execution paths and of propositional logic relations between the elements in the activity. Hence, in this chapter a transformation is presented that ensures this condition.

As a prerequisite for addressing **Problem Statement 2**, this chapter contributes to the constructive portion of this thesis. While its main contribution is the removal of redundant elements in order to generate textual representations (contribution IV), this method is constructed in the most generalizable way possible. As a result the method is more generally applicable. Consequently, the method ex-

tends beyond the formulated problem statements of this thesis (contribution VI).

#### AUTHOR CONTRIBUTIONS

The author list includes Martin Beckmann, Vanessa N. Michalke, Andreas Vogelsang, and Aaron Schlutter. The author of this thesis was the lead author of the publication. He wrote the majority of the article and conceived the method and developed it with support of Aaron Schlutter. Vanessa N. Michalke programmatically implemented the method and collected the data for the evaluation. Andreas Vogelsang provided advice on the research design. Vanessa N. Michalke, Andreas Vogelsang, and Aaron Schlutter all contributed to the writing of the paper.

# Removal of Redundant Elements within UML Activity Diagrams

Martin Beckmann, Vanessa N. Michalke, Andreas Vogelsang, Aaron Schlutter  
Technische Universität Berlin, Germany  
{martin.beckmann, vanessa.michalke, andreas.vogelsang, aaron.schlutter}@tu-berlin.de

**Abstract**—As the complexity of systems continues to rise, the use of model-driven development approaches becomes more widely applied. Still, many created models are mainly used for documentation. As such, they are not designed to be used in following stages of development, but merely as a means of improved overview and communication. In an effort to use existing UML2 activity diagrams of an industry partner (Daimler AG) as a source for automatic generation of software artifacts, we discovered, that the diagrams often contain multiple instances of the same element. These redundant instances might improve the readability of a diagram. However, they complicate further approaches such as automated model analysis or traceability to other artifacts because mostly redundant instances must be handled as one distinctive element. In this paper, we present an approach to automatically remove redundant *ExecutableNodes* within activity diagrams as they are used by our industry partner. The removal is implemented by merging the redundant instances to a single element and adding additional elements to maintain the original behavior of the activity. We use reachability graphs to argue that our approach preserves the behavior of the activity. Additionally, we applied the approach to a real system described by 36 activity diagrams. As a result 25 redundant instances were removed from 15 affected diagrams.

## I. INTRODUCTION

Due to its many advantages [1], model-driven engineering has become a widely applied approach in the development of systems [2]. One of our industry partners (Daimler AG) uses UML2 activity diagrams [3] to specify the functions of systems. Activity diagrams are behavioral diagrams used to create graphical models of stepwise workflows. They are among the types of models, which are regarded beneficial in requirements engineering [4]. A widely applied use of activity diagrams and graphical models in general is to utilize them for communication purposes [5], [6]. Hence, the diagrams are created with a focus on readability and understandability. This is achieved by prioritizing layout aspects of the diagram, since a proper layout is an important factor for understanding the diagrams [7]. As a result, the created diagrams are not catered to be processed by automatic approaches in following stages of development. One of the phenomena that impede automation are multiple instances of the same element within the activity diagrams. Some of these redundant elements are created intentionally [8] to improve certain aspects of a diagram such as the structure and hence the readability. Other redundant elements arise unintentionally since multiple persons are involved in creating the diagrams. Nonetheless, existing redundant elements complicate possible approaches

of automation. For instance, in requirements engineering activities need to be accompanied by textual representations [9]. As a result, a well-structured requirements document should reflect, which executions are possible and necessary in an activity. This relies on the propositional assertions modeled in the activity. To apply simplifications on propositional logic relations (e.g., extract a propositional logic normal form), it is necessary to know, which elements are actually the same. This is guaranteed by a redundancy-free version. Also, for an effective derivation of test cases from activities, path coverage has to be considered. If there are redundant elements, there may be unnecessary paths and hence more test cases are required [10]. In general, checking for path coverage is easier without redundant elements since only unique paths are considered. Other than that, redundancy makes traceability and keeping derived artifacts consistent more difficult.

This paper presents a transformation to remove redundant *ExecutableNode* elements contained within an activity. The removal is achieved by merging all instances of a redundant element into a single instance. In order to preserve the original behavior, new *ControlNodes* are added in the activity. These new *ControlNodes* are connected to the merged element as well as its predecessors and successors. We show that these model transformations preserve the behavior of the original activity by comparing their reachability graphs. We furthermore report on the application of this approach to a set of activity diagrams used to specify a real system from our industry partner. This evaluation shows that redundant elements are common in real activity diagrams and that our approach is able to remove them without blowing up the complexity of the diagrams.

The paper's structure is given in the following. The next section provides details on the situation we encountered at our industry partner. The third section discusses related work on the subject of behavior preserving transformations and of dealing with redundancy in graphical models. In the fourth section, we present the behavior-preserving transformation that removes redundant elements. Section V shows why the transformation preserves the behavior of the activity. Section VI introduces special situations, where less *ControlNodes* are needed for the transformation. In Section VII we analyze activities supplied by our industry partner and present results on applying the transformation on them. Section VIII presents the limits of the approach. The last section concludes this work and gives an outlook on future work.

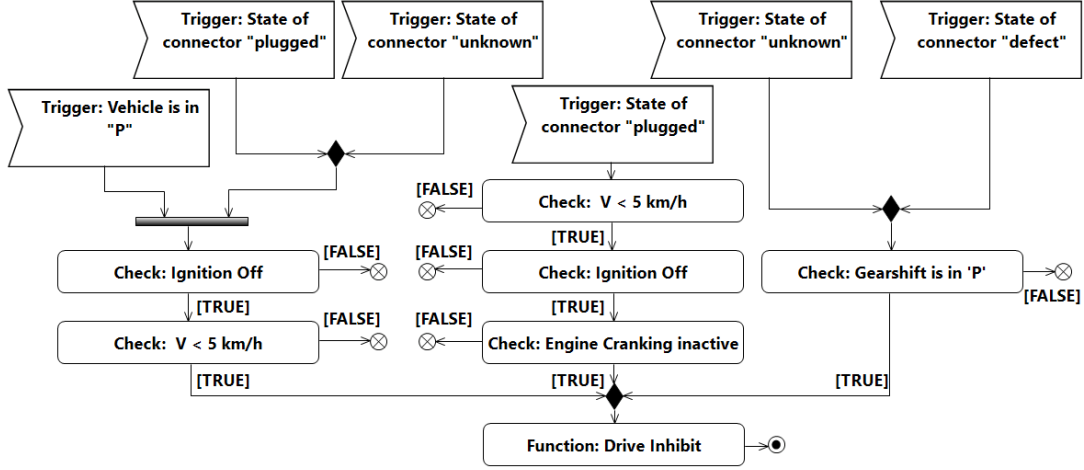


Fig. 1: Exemplar activity diagram containing redundant elements

## II. BACKGROUND

In this section we use an activity diagram provided by our industry partner to show how their diagrams are used and interpreted. In addition, we present our notion of redundancy in activity diagrams.

### A. Activity Diagrams Syntax

Our industry partner uses UML activity diagrams as a first step of specifying a new function of a system. An exemplar activity diagram<sup>1</sup> is displayed in Fig. 1. It describes the function's activation and deactivation.

As such, the activity diagram contains a combination of triggers and checks for conditions that must be fulfilled to activate the function. This type of description is similar to Firesmith's proposal for formulating textual natural language requirements [11]. For triggers, the *AcceptEventAction* element is used. Checks are modeled as *Action* elements. If the condition of a check is not fulfilled, the flow ends (*FlowFinal*). The triggers and checks are connected by *ControlNodes* such as *JoinNodes* and *MergeNodes*. *JoinNodes* act as synchronization points and can be interpreted as AND operators in terms of propositional logic. *MergeNodes* represent OR operators. Once the actual functionality of the function is executed, *ActivityFinal* elements designate the end of an activity.

### B. Activity Diagram Semantics

We interpret the semantics of activities as Petri net like graphs as suggested by the original UML specification version 2.5 [3, p. 283]. As such, we assume that each *ExecutableNode* executes as soon as a token is placed on that node (by transition or by occurrences of events). Also, we assume that the execution time of the nodes is infinitely fast. This interpretation is related to *requirements-level semantics* for the activity diagram defined by Eshuis and Wieringa [12] and is also used by our industry partner. Furthermore, *ControlNodes* forward

tokens instantly if possible. Hence, tokens can be forwarded by multiple *ControlNodes* in one step. Events that execute *AcceptEventActions* of the activity, produce new tokens within the executing activity (i.e., the property *isSingleExecution* is true). Also, we assume that two tokens at an *ExecutableNode* cause two concurrent executions of the *ExecutableNode* within the same step (i.e., the property *isLocallyReentrant* is true).

### C. Redundant Elements in Activity Diagrams

In this paper, we are only interested in redundant elements within one diagram and not across different diagrams. The activity diagram, shown in Fig. 1, contains four redundant elements, each having two instances in the diagram. The triggers *State of connector "plugged"* and *State of connector "unknown"* both appear two times. Similar, the two checks *V < 5 km/h* and *Ignition Off* also appear twice in the diagram. Elements are considered as redundant elements if they have the same name and the same type (e.g. *AcceptEventAction*). Thus, the considered elements are exact copies of each other apart from their placement within the diagram and their connection to other elements, which makes these elements **Type A Clones** according to Störrle's classification [13].

While this duplication of the same element increases the number of elements in the diagram, it may also increase the comprehensibility of the diagram. For instance, the duplicated elements in Fig. 1 allow the visual separation of three distinct possibilities, that lead to the function's activation.

In a previous work, we defined and analyzed different types of quality issues that arise when activity diagrams are used in requirements documents for the specification of functions [14]. One of the quality issues, we identified, are redundant elements. In that study, we found redundant elements in more than 40% of the examined diagrams, which shows that this is a common phenomenon. On the other hand, developers rated the appearance of redundant elements as one of the least severe quality issues. However, this work solely focuses on enabling the use of the diagrams for automation rather than creating an alternative view for existing diagrams.

<sup>1</sup>The displayed activity diagram was slightly modified to incorporate more sophisticated situations.

### III. RELATED WORK

Since our approach transforms models in a behavior preserving way, it is related to refactoring [15]. In contrast to classical refactoring, our aim is not to improve the design but to facilitate the processing by automated techniques.

Refactoring of UML models has been covered by a number of publications [16]. Focal point of their research is the UML class diagram as it is the most used UML diagram [17]. Among others, examples for refactorings of class diagrams are presented in [18], [19], [20], [21]. Another type of diagram that has received attention in relation to refactoring are statecharts [18], [21]. For activity diagrams, two refactoring operations are described in [21], namely *Make Actions Concurrent* and *Sequentialize Concurrent Actions*. As these names of the operations indicate, they are not suitable to deal with redundant elements. A more extensive review on refactoring UML models can be found in [22]. An approach of detecting semantically equivalent modeling concepts for structurally different models is described in [23].

Besides the transformation of models, redundancy in UML models has also been a topic in research, although the focus is mainly placed on the detection of redundancy (see [24] for a list of approaches). For Petri nets, as a basis for activity diagrams, the elimination of redundant control places while keeping a Petri net live is described by Uzam et al. [25]. In addition to presenting an approach to detect clones in models, Störrle gives an example of a transformation that removes recurring fragments of activities by factoring them into independent activities [13]. The main rationale behind these refactoring proposals is to increase the maintainability of models and reduce the risk of inconsistent changes. Our work, in contrast, deals with the removal of redundant elements within a single activity for the purpose of facilitating their processing by automatic approaches. This also includes elements that UML defines as “integral parts of a diagram” (such as *DataFlowNodes*), which Störrle calls *loophole clones* and for which his refactoring approach does not work. It is also mentioned, that there are tools distinguishing between internal representation (the activity itself) and an external visual representation (the activity diagram). Most contemporary tools enforce a one-to-one correspondence between these two representation types. This simplifies, the handling of copy/paste operations [13]. As a consequence, using an element of the internal representation multiple times in the external representation is not possible. A tool, that does not enforce one-to-one correspondence would allow for proper readability while still allowing for automatic approaches. Nonetheless, even if such a tool is used, the modeler must still be aware of this capability and is required to consider this fact during model creation. Hence, a tool-independent approach is needed, that takes into account the modeling-process and its challenges.

Activities can be the source for a number of possible applications. Among others they are used to automatically

generate textual specification documents [26], source code [27] and test cases [28].

### IV. ELIMINATION OF REDUNDANT ELEMENTS

#### A. Transformation

To remove redundant *ExecutableNode* elements from an activity, we propose a transformation that consists of three steps. All steps have to be performed for every redundant element in an activity. The first step adds a new element, that represents all instances of the redundant element. The second step adds *ControlNodes* to the activity. The predecessors and successors of the instances and the added element are connected with the *ControlNodes* by *ControlFlows*. Lastly, all instances of the considered redundant element are removed from the activity. As a result the element added in the first step remains as a single instance of the redundant element.

The necessary *ControlNodes* are *ForkNodes*, *JoinNodes*, *MergeNodes* and *DecisionNodes*. A single *MergeNode* is added as a predecessor to the remaining element. A single *DecisionNode* is added as a successor to the remaining element. For each instance of the redundant element one *ForkNode* and one *JoinNode* is added. The *ForkNodes* are predecessors to the added *MergeNode*. The predecessor of each *ForkNode* are the predecessors of the original instances. The *JoinNodes* are successors of the added *DecisionNode*. The successors of each *JoinNode* are the successors of the original instances. Each of the added *ForkNodes* has an outgoing edge to an added *JoinNode*. Thereby, the *ForkNode*, which is added as the predecessor to one instance is connected to the *JoinNode*, which is added as the successor to the same instance. Since there are no guards on the outgoing edges of the *DecisionNode*, an incoming token is forwarded to a *JoinNode* with a token present [3, p. 373, p. 387].

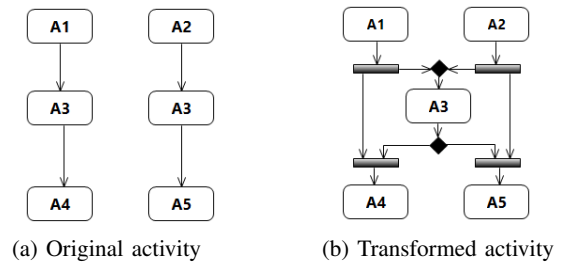


Fig. 2: Example of the transformation

Fig. 2 shows an activity diagram fragment before (Fig. 2a) and after (Fig. 2b) the transformation. The depicted activity has the redundant element A3 with the two redundant instances A3<sub>1</sub> and A3<sub>2</sub>. The instances are denoted with indices for distinction. The Actions A1, A2, A4 and A5 can be any *ExecutableNode* (e.g. *Actions*, *AcceptEventActions*) or multiple *ExecutableNodes* or *ControlNodes*. In case A1 executes, an execution of A3 follows. Because of the execution of A1, there is a token present at the *JoinNode* before A4. The token produced by A3 is forwarded to this *JoinNode*. Thus A4 executes. In case both A1 and A2 are executed, A3 is executed

two times and hence produces two tokens, which lead to the execution of A4 and A5. In both cases, it is the same behavior as before.

### B. Normal Form

The transformation is applicable if all instances of the redundant elements have a single predecessor and a single successor. For this purpose, we denote an activity with redundant elements, where each of its instances has exactly one predecessor and one successor, to be in a normal form. If this is not the case, further *ControlNodes* are added to make sure this requisite is fulfilled.

An activity is not in the normal form if one of its instances is missing a predecessor or a successor. If the predecessor is missing, an *InitialNode* is added as a predecessor [3, p. 376]. If the successor is missing, a *FlowFinal* is added as a successor since the flow of tokens ends after the *ExecutableNode*. In Fig. 3a a situation is displayed, where one element does not have a predecessor and one element does not have a successor. Fig. 4a shows the corresponding situation with an added *InitialNode* and *FlowFinal*.

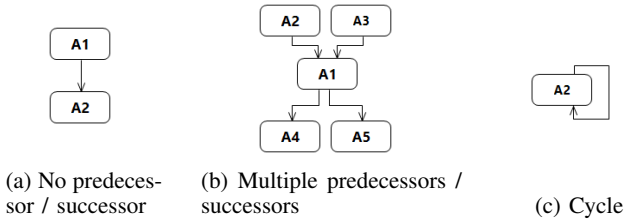


Fig. 3: Situations without normal form

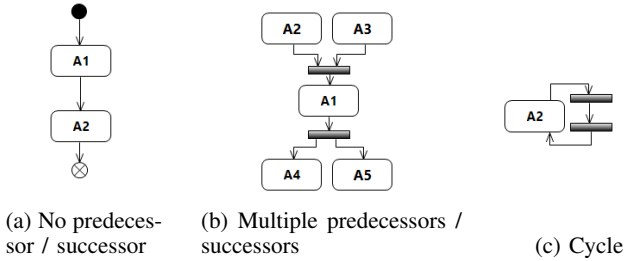


Fig. 4: Situations of Fig. 3 in normal form

In addition to missing predecessors or successors, there might be instances with more than one predecessor or successor. If there are multiple predecessors, a *JoinNode* is added to make the implicit *Join* to an explicit *Join* [3, p. 401]. After the transformation, this results in a *JoinNode* with an outgoing edge to the respectively added *ForkNode* before the remaining element. If there are multiple successors, a *ForkNode* is added to make the implicit *Fork* to an explicit *Fork* [3, p. 401]. After the transformation, this results in a *ForkNode* with an incoming edge from the respectively added *JoinNode* after the remaining element. An example of the described situation is displayed in Fig. 3b. The corresponding, for Action A1 resolved situation is shown in Fig. 4b.

An instance of a redundant element can also be a part of a cycle. In case the instance is its own predecessor and successor, it is necessary to add *ControlNodes* as new predecessors and successors. As a predecessor a *ForkNode* is added and as a successor a *JoinNode* is added. In Fig. 3c such a situation is displayed. Fig. 4c shows the corresponding situation with an added *JoinNode* and *ForkNode*. It is also possible to add a *DecisionNode* and a *MergeNode* or any other combination of *ControlNodes*, as these *ControlNodes* only have one incoming and one outgoing edge. However a *ForkNode* and a *JoinNode* can be merged, with the respective *JoinNode* and *ForkNode*, that are added by the transformation.

By using *ControlNodes* with one incoming and outgoing edge, ultimately every combination of predecessors and successors of the redundant elements can be converted to the normal form.

### C. Number of additional elements

The number of necessary additional *ControlNodes* and *ControlFlow* edges depends on the number of redundant elements and on how many instances are part of each redundant element. There is a new *ForkNode* and a new *JoinNode* for each instance. Also, there is one additional *MergeNode* and one additional *DecisionNode* for each redundant element. Besides, there might be *ControlNodes* necessary to ensure, that each instance has a single predecessor and successor. Thus, the number of new *ControlNodes* in an activity results in:

$$\#NewControlNodes = \sum_{i=1}^m (2 * n_i + 2) + c \quad (1)$$

The variable  $m$  denotes the number of redundant elements,  $n_i$  denotes the number of instances in each redundant element and  $c$  denotes the number of *ControlNodes* needed for the normal form. Since, there is a maximum of two *ControlNodes* needed for every instance to create the normal form, there is a linear relation between the number of redundant elements and the additionally needed *ControlNodes*.

Using the same notation and, additionally, the variable  $d$  as the number of needed edges, the number of new *ControlFlow* edges in an activity results in:

$$\#NewControlFlowEdges = \sum_{i=1}^m (3 * n_i + 2) + d \quad (2)$$

For every instance of a redundant element, there need to be three additional edges (two outgoing edges of the added *ForkNode*, one incoming edge of the added *JoinNode*). For every redundant element two edges are needed as the outgoing edge of the *MergeNode* and the incoming edge of the *DecisionNode*. Additionally, there is a maximum of two edges needed for each instance of a redundant element, to create the normal form. Hence, there is also a linear relation between the number of redundant elements and the additionally needed *ControlFlow* edges. The linear relations for the number of nodes and edges are important, since an automated processing might be impaired otherwise.



#### D. Resulting Structure

The usage of a single predecessor and a single successor results in a single-entry single-exit structure for the transformed part. As a consequence everything before and after the remaining element stays unchanged. The principle of compositionality applies. This means, that the behavior of the activity remains the same, if the behavior of the changed part remains the same. Hence, to show the preservation of the behavior, it is sufficient to show, that the behavior of the transformed part stays the same.

#### V. PRESERVATION OF BEHAVIOR

To show that the transformation preserves the behavior expressed in an activity diagram, we compare the flow of tokens in the underlying semantic model (see Section II). Reachability graphs (RG) [29] represent this flow of tokens in a network depending on the executed actions. Hence, we use reachability graphs as a means to show, that the behavior of the activity, before and after applying the transformation, is still the same. In this chapter, we briefly introduce reachability graphs. Additionally, we argue why the comparison of the RGs of the activities is suitable to show the preservation of the behavior. Subsequently, we propose a way to derive RG from activities. In the subsection after that, we show how to compare two activities by using RGs.

##### A. Reachability Graphs

We construct the reachability graph RG for an activity  $A$  as:

$$RG(A) = (M(V), E) \quad (3)$$

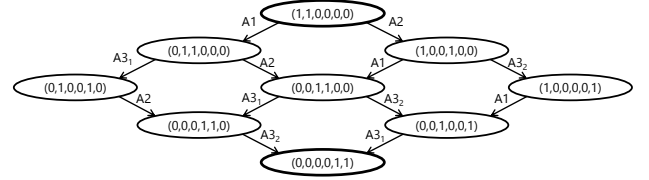
$V$  is the set of *ActivityNodes* contained in the activity  $A$ .  $M(V)$  is the set of distributions of tokens to the *ActivityNodes*. Thus, every node  $m \in M(V)$  in the  $RG$  represents a distribution of tokens within the activity  $A$ . Every element of  $M(V)$  is a  $|V|$ -tuple, where each entry represents the number of tokens at every *ActivityNode* after a sequence of executions.  $E$  is the set of directed edges of the  $RG$ . The edges represent the execution of an *ExecutableNode*, which leads to a new distribution of tokens.

The *initial distribution*  $m_0 \in M(V)$  represents the distribution of tokens at the beginning of execution. Its node in the RG has no incoming edges. The *initial distribution* depends on the events that might occur at the beginning and on existing *InitialNodes*. There may be multiple different *initial distributions*, which each result in different RGs. The *final distribution*  $m_n \in M(V)$  represents the distribution of tokens in the activity, where no more nodes are left to be executed. This is also the case as soon as one token reaches an *ActivityFinal*. Its node in the RG has no outgoing edges. Each RG has only one *final distribution*.

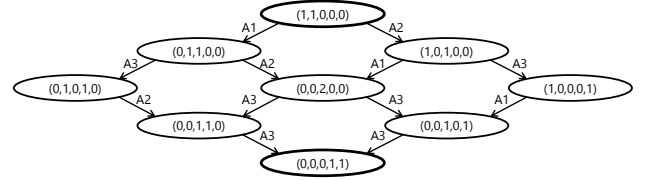
A *sequence of executions* is a sequence of edges  $(e_1, \dots, e_n)$  in the reachability graph RG, which starts at the *initial distribution* and ends at the *final distribution*. Hence, a *sequence of executions* represents a possible order of executed *Actions* in

the activity that lead from the *initial distribution* to the *final distribution*.

Fig. 5a shows the RG of the activity presented in Fig. 2a. The RG of the transformed activity from Fig. 2b is displayed in Fig. 5b. In both cases, it is assumed that the *initial distribution* results from the events executing  $A1$  and  $A2$  simultaneously. Also, the distributions in both figures only incorporate *ExecutableNodes*.



(a) RG ( $A1, A2, A3_1, A3_2, A4, A5$ ) for the original activity



(b) RG ( $A1, A2, A3, A4, A5$ ) for the transformed activity

Fig. 5: RGs for the activities in Fig. 2

While the distribution of tokens in Fig. 5a is represented by a 6-tuple, the distribution in Fig. 5b is represented by a 5-tuple. This results from the different number of *ExecutableNodes* in the two activities, since the two redundant *Actions*  $A3_1$  and  $A3_2$  were replaced by  $A3$ .

An RG contains all possible *sequences of executions* of an activity for a given *initial distribution*. As a consequence, we conclude that if the same sequences lead to the same distribution of tokens in an activity, then the behaviors of the activities are same. As a result, the comparison of two RGs of the respective activities shows the preservation of the behavior for a given *initial distribution*.

##### B. Generating Reachability Graphs from Activity Diagrams

RGs are generated for a chosen *initial distribution*. Hence, the first step is to decide how many tokens are initially placed on each *ActivityNode*. The resulting distribution of tokens is the first node (the *initial distribution*) of the RG.

From the *initial distribution*, the RG is constructed step by step. The underlying algorithm is basically the same as for a Petri net. For every entry in the current distribution, which has at least one token, the token is transferred from the corresponding *ActivityNode* to its successor in the activity. This results in a new distribution in the RG, which is connected by an incoming edge to the previous distribution in the RG. Although the *ControlNodes* do not hold tokens, they are included as entries in the distributions. This is necessary, since they may change the number of tokens. As a result the number

of tokens of a new distribution depends on the type of the executed *ActivityNode*. If the current *ActivityNode* is an *ExecutableNode*, every token is forwarded to the successor after the *ExecutableNode* is executed (assuming that there are no implicit *ControlNodes*). The different types of *ControlNodes* on the other hand all show a different behavior towards the number of tokens in the activity. Hence, every *ControlNode* needs to be considered differently. In the following,  $t(v)$  denotes the number of tokens at a certain *ActivityNode*  $v \in V$ .

**MergeNode.** *MergeNodes* forward tokens from multiple incoming *ActivityEdges*. As such, they act as OR-connections between the predecessors. As a consequence, for a given distribution  $(\dots, t(v), \dots, t(v'), \dots)$ , where  $v \in V$  is the *MergeNode* and  $v' \in V$  is a successor, the distribution  $(\dots, t(v) - 1, \dots, t(v') + 1, \dots)$  is added as a node in the RG.

**ForkNode.** *ForkNodes* pass a single token to each outgoing *ActivityEdge* for each token on the incoming *ActivityEdge*. For a given distribution  $(\dots, t(v), \dots, t(v_1), \dots, t(v_n), \dots)$ , where  $v \in V$  is the *ForkNode* and  $v_1, \dots, v_n \in V$  are the following nodes on the outgoing *ActivityEdge* of the *ForkNode*, results the distribution  $(\dots, t(v) - 1, \dots, t(v_1) + 1, \dots, t(v_n) + 1, \dots)$ .

**JoinNode.** *JoinNodes* act as AND-connections as they only forward a single token on their outgoing *ActivityEdge*, if there is one token present at each incoming *ActivityEdge*. For a given distribution  $(\dots, t(v), \dots, t(v'), \dots)$ , where  $v \in V$  is the *JoinNode* and  $v' \in V$  is the successor, if there are  $n \in \mathbb{N}$  incoming *ActivityEdges*, where each incoming *ActivityEdge* has a single token present, the distribution  $(\dots, t(v) - n, \dots, t(v') + 1, \dots)$  is created.

**DecisionNode.** *DecisionNodes* forward a single token to an applicable outgoing *ActivityEdge* if there is a token on the incoming *ActivityEdge*. If there is more than one *ActivityEdge* applicable, the token only traverses one of the *ActivityEdges* [3, p. 388]. Since there are different RGs depending on the decisions, there has to be a new RG for every possible decision and not just a new distribution. For a given distribution  $(\dots, t(v), \dots, t(v_1), \dots, t(v_i), \dots, t(v_n), \dots)$ , where  $v \in V$  is the *DecisionNode*,  $v_1, \dots, v_n \in V$  are successors of the *DecisionNode* and  $v_i$  is the node, that accepts the token offered by the *DecisionNode*, follows the new distribution  $(\dots, t(v) - 1, \dots, t(v_1), \dots, t(v_i) + 1, \dots, t(v_n), \dots)$ .

### C. Comparison of Activity Diagrams

For a complete comparison of the behavior of two activities, one would need to compare all possible RGs of both activities. Different RGs arise from different *initial distributions* and non-deterministic *DecisionNodes* [3, p. 387]. Due to the unlimited number of tokens that may be placed in the *initial distribution*, there is an infinite number of possible RGs. Therefore, in general, it is not possible to compare all RGs. We argue that it is sufficient to compare just the RGs with all possible combinations of *initial distributions*, where each suitable *ActivityNode* has either no token at all or just one token. The reason is that all combinations of zero or one token represent all possible flows in the activity. This rationale ignores the fact that there might be structures in an activity that

require a certain number of tokens. Suitable *ActivityNodes* are those that might have a token at the start of the execution of an activity (*InitialNodes* and *AcceptEvenActions*). For  $n$  suitable *ActivityNodes*, this results in  $2^n - 1$  different RGs for different *initial distributions*. The distribution of no tokens at all does not yield any information regarding the behavior.

These RGs are needed for the comparison of the behavior of two activities. The actual comparison of two RGs is done with two RGs representing the same situation, i.e., the same *initial distribution* and the same decisions made. One criterion for two RGs to be equivalent is that their *sequences of execution* are equivalent. This means, that every *sequence of execution* in the RG of the original activity has an equivalent *sequence of execution* in the RG of the transformed activity. As the *ControlNodes* do not hold tokens and do not execute operations other than manipulating the flow of tokens, they must not be considered during the comparison of two RGs. Note that there are redundant sequences of executed *Actions* in the RGs because of the redundant elements.

Besides the *sequences of execution*, the equality of the individual distributions is the second criterion that must be fulfilled. Otherwise, the behavior is not the same if the same sequences lead to different distributions. The transformed activity contains less entries in the distributions of the RG than the original activity (see subsection V-A) because of the removed redundant elements. Thus, the sum of tokens of the redundant elements equals the number of tokens of the remaining elements. For the distribution  $(y_i, \dots, y_{m-(n-i)})$  of the transformed activity an equivalent distribution  $(x_1, \dots, x_i, \dots, x_n, \dots, x_m)$  of the original activity, given redundant instances  $x_i$  to  $x_n$ , can be identified by:

$$y_k = \begin{cases} x_k & \text{if } k = [1, i - 1] \\ \sum_{l=i}^n x_l & \text{if } k = i \\ x_{k+(n-i)} & \text{if } k = [i + 1, m - (n - i)] \end{cases} \quad (4)$$

Because of the single-entry single-exit structure of the transformation (see subsection IV-A), it is only necessary to compare the changed part of the activity before and after the transformation. Since the transformation introduces pairs of *ForkNodes* and *JoinNodes* for the respective predecessors and successors of the original instances, the forwarding of tokens by the introduced *DecisionNodes* are deterministic and hence do not require additional RGs. The deterministic behavior results from the fact that it is always clear to which *JoinNode* a token is forwarded in each step. In the example Fig. 2a, there are two suitable *ActivityNodes* for the *initial distribution*. Thus, three pairs of RGs need to be compared.

The *sequences of executions* of the RG in Fig. 5a are the following:

- |  |  |
|--|--|
| 1) A1, A3 <sub>1</sub> , A2, A3 <sub>2</sub> | 4) A2, A1, A3 <sub>1</sub> , A3 <sub>2</sub> |
| 2) A1, A2, A3 <sub>1</sub> , A3 <sub>2</sub> | 5) A2, A1, A3 <sub>2</sub> , A3 <sub>1</sub> |
| 3) A1, A2, A3 <sub>2</sub> , A3 <sub>1</sub> | 6) A2, A3 <sub>2</sub> , A1, A3 <sub>1</sub> |

The *sequences of executions* of the RG in Fig. 5b are:

- |                   |                   |
|-------------------|-------------------|
| 1) A1, A3, A2, A3 | 4) A2, A1, A3, A3 |
| 2) A1, A2, A3, A3 | 5) A2, A1, A3, A3 |
| 3) A1, A2, A3, A3 | 6) A2, A3, A1, A3 |

Since Action  $A3_1$  and  $A3_2$  in the original activity in Fig. 2a are redundant instances of the same redundant element, the *sequences of executions* 2) and 3) as well as 4) and 5) are the same. By removing the redundant *sequences of executions* and comparing the remaining ones, it follows that both RGs contain the same *sequences of executions*.

Considering the criteria defined in Equation 4 for the equality of the distributions, the distributions are equal as well. As a result, the RGs in Fig. 5 are isomorphic. The same holds for the RGs generated by using the other two *initial distributions*. Hence, for the assumed semantics, the transformation preserves the behavior if it is applied to an activity containing one redundant element consisting of two instances. The transformation can also be applied to activities with more than one redundant element and with more than two instances. This is based on the fact that the transformation can be applied in any order. From this follows that the removal of multiple instances can be conducted by applying the transformation to only two instances each time until there is one instance left. Additional *ControlNodes* resulting from the consecutive application of the transformation can be merged.

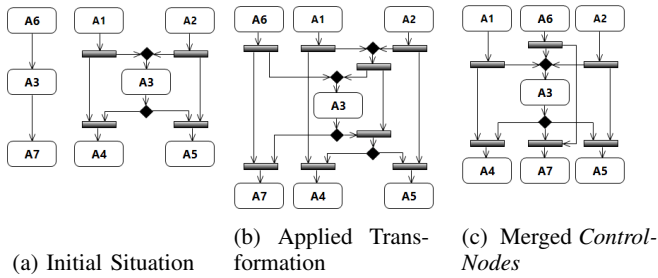


Fig. 6: Transformation of a redundant element with three instances

The consecutive application of the transformation is shown in Fig. 6. In the initial situation in Fig. 6a, there is a redundant element with two instances. The structure of the instance on the right hand side, results from a previous application of the transformation on two instances. If the transformation is applied in this situation the *ExecutableNode* A6 and the added *MergeNode* are used as the predecessors and the *ExecutableNode* A7 and the added *DecisionNode* are used as successors. This results in the activity displayed in Fig. 6b. This structure can be simplified to the structure displayed in Fig. 6c without changing the flow of tokens. Since the added *MergeNode* and *DecisionNode* are now the new predecessor and successor, this procedure can be repeated if there are further instances.

## VI. SPECIAL SITUATIONS

Besides the presented transformation, there are special situations where the removal of the redundant elements is possible

using less additional *ControlNodes*. Three examples are shown in Fig. 7.

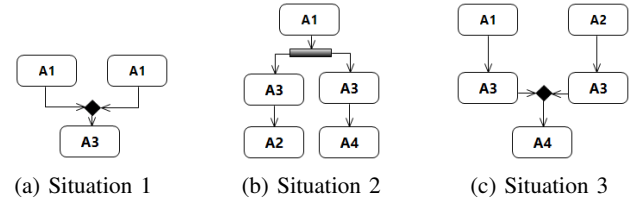


Fig. 7: Three special situations with redundant elements

The activity in Fig. 7a contains two redundant elements. Both elements do not have a predecessor. The activity in Fig. 7b contains two redundant elements, which have a common predecessor and distinct successors. The activity in Fig. 7c contains two redundant elements, which have distinct predecessors and a common successor.

Their respective activities without redundant elements are depicted in Fig. 8.

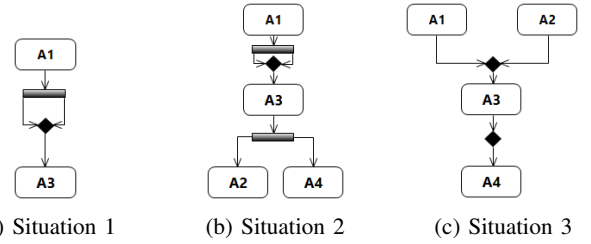


Fig. 8: Three special situations without redundant elements

In all of these situations there is only one of the two redundant instances left. Compared with the generic transformation introduced in subsection IV-A, less additional *ControlNodes* and *ControlFlow* edges are needed. Hence, the resulting number of *ControlNodes* and of *ControlFlow* edges in Equation 1 and in Equation 2 respectively are upper limits.

The preservation of behavior for these activities results from the fact that the missing predecessors and successors lead to structures that do not change the flow of the tokens. For instance, there are *ControlNodes* without incoming edges or *ControlNodes* with a single incoming and a single outgoing edge. We additionally verified the preservation of behavior by constructing the RGs for these activities. The resulting RGs are equivalent for the necessary distributions.

### A. Introductory Example Revisited

If the presented approach is applied to the introductory example in Fig. 1, this results in the activity diagram displayed in Fig. 9. As the activity is meant as a source for further automated approaches, the activity diagram is displayed to illustrate the applied transformations. To increase the readability, we left out the *FlowFinal* elements and not all of the implicit connections are depicted explicitly. As there are four redundant elements with two instances each, four elements are

removed. For the redundant triggers, the transformation for situation 1, presented in Section VI is applied. For the redundant checks the generic transformation is applied. In contrast to the original activity, the number of *ControlNodes* increases and intersecting edges appear. This results in decreased readability, which makes the activity harder to understand. Since the aim of the transformation is to improve applicability of automated approaches, the decreased readability is not an issue within the scope of this work. Hence, we propose to use the redundancy-free activity as a parallel artifact.

## VII. EVALUATION

To evaluate the applicability of our approach in activity diagrams created in practice, we applied the approach to the activity diagrams of a system of an industry partner. The system's functions were specified by a total of 36 activity diagrams (containing between 9 to 28 *ExecutableNodes*). Each activity describes a function of a system, which is responsible for charging the high-voltage batteries of Plug-in Hybrid Electric Vehicles and Battery Electric Vehicles. As such the system contains functions that are relevant for safety as well as for usability. The aim of the evaluation is to answer the following questions:

- **RQ1:** How many redundant elements appear in activities of a real system?
- **RQ2:** Is our transformation approach applicable to every situation in the activities of a real system?
- **RQ3:** How many additional elements are introduced when applying the transformations?
- **RQ4:** How many of the special situations occur in the activities of a real system?

### A. Implementation

The original data of our industry partner was supplied as an Enterprise Architect project file. To apply the transformation, two steps were required to prepare the data. The first step is to convert the project file (*.eap*) to a *.uml* file. The conversion was done automatically by a self-written converter. Our implementation of the transformation approach is realized using *.uml* files because, in contrast to, e.g., the *.eap* format, the *.uml* data format is aligned with the UML specification. Hence, it is only necessary to adjust the converter if a different data source is used in future. In the second step, the resulting *.uml* file is edited manually. This is necessary if our converter encountered situations that it could not handle. Such situations may result from deviations between the Enterprise Architecture data model and the *.uml* data model. Another reason for the manual adjustments was to correct the use of a number of *ControlNodes* originating from developer mistakes and misunderstandings (e.g. *MergeNodes* and *DecisionNodes* as well as *JoinNodes* and *ForkNodes* were sometimes mixed up because they look the same). The transformation itself was implemented to work on the resulting *.uml* files.

TABLE I: Extent of redundant elements in the analyzed system.

Finding	#Diagrams	Ratio
Total activity diagrams	36	100%
Containing redundant elements	15	42%
Containing 1 redundant element	8	19%
Containing 2 redundant elements	5	17%
Containing 3 redundant elements	2	6%
Finding	#Red. Elements	Ratio
Total redundant elements	24	100%
Containing 2 instances	23	96%
Containing 3 instances	1	4%

TABLE II: Results of applying the transformation

Finding	Number
Removed <i>ExecutableNodes</i>	25
Added <i>ControlNodes</i> max.	146
Added <i>ControlNodes</i> min.	111
Generic Transformation	18
Situation 1	4
Situation 2	0
Situation 3	3

### B. Study Results

To answer the first question, we analyzed the activities towards the number of occurrences of redundant elements. The detailed results are displayed in Table I.

Out of the 36 diagrams, we found 15 diagrams containing redundant elements. The 15 diagrams contain 24 elements that appear multiple times in each activity diagram. Of these 15 diagrams, there are 2 diagrams each containing 3 redundant elements with two instances. Another 5 diagrams contain 2 redundant elements. Out of these 5 diagrams 4 diagrams have redundant elements with two instances each. The fifth diagram contains a redundant element with three instances as well as one with two instances. The remaining 8 diagrams contain only one redundant element with two instances each.

The transformation was applicable to all provided activities. There is no constellation, where the transformation would not preserve the behavior. An analysis of the results of the applied transformations is shown in Table II.

The removal of the 23 redundant elements with two instances and the one redundant element with three instances results in an overall of 25 removed *ExecutableNodes*. When only applying the generic transformation, a total of 146 *ControlNodes* were added (*Added ControlNodes max.*) to the activities. However, when also using the smaller transformations for the special situations explained in section VI, the generic transformation only had to be applied 18 times. Situation 1 was applicable four times and situation 3 was applicable three times. The special situation 2 did not occur. By utilizing the

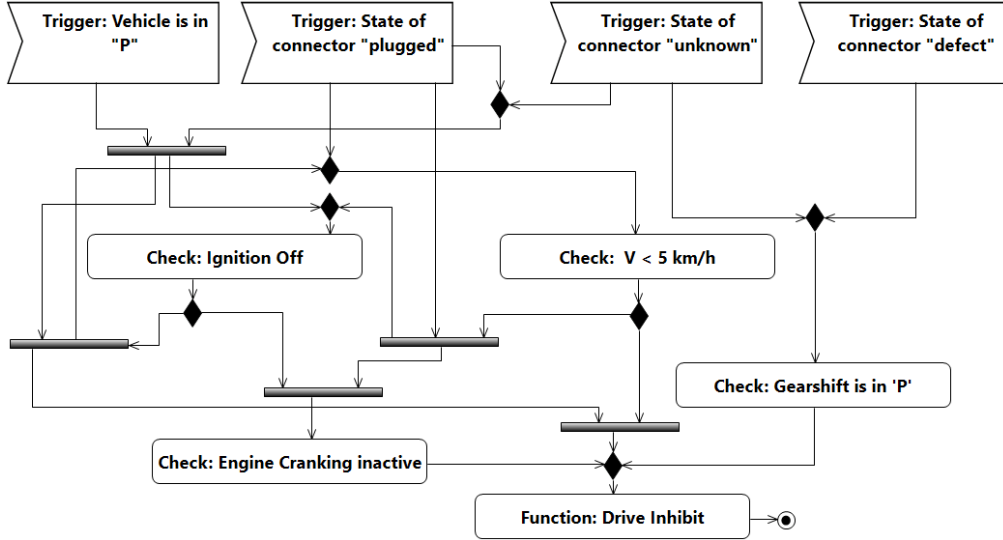


Fig. 9: Introductory example without redundant elements

transformations of the special situations, only 111 additional *ControlNodes* had to be added.

It has to be noted, that we only examined one system of one single industry partner. As a result, the generalizability of our findings is limited in regard to whether the approach is always applicable and whether the resulting numbers are representative.

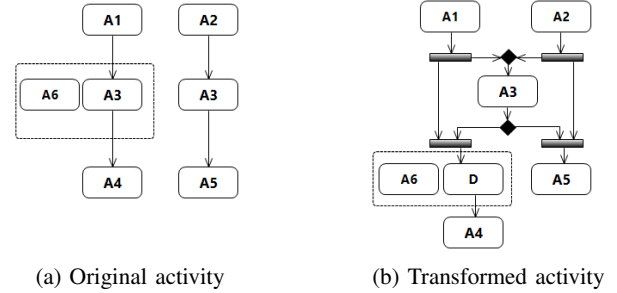
#### VIII. LIMITATIONS OF THE APPROACH

The presented approach is restricted to our interpretation of the semantics of an activity. If different semantics are used, the transformation might no longer preserve all aspects of the behavior. Assuming a semantic where every *ExecutableNode* has its own individual execution time and events can occur at any time, this might lead to tokens overtaking one another. Hence, the order in which the actions are executed is no longer the same. Still, the same *Actions* are executed the same number of times. A possible sequence of executions of the activity in Fig. 2a is shown in 1). A possible deviating sequence of executions of the redundancy-free activity in Fig. 2b is shown in 2).

- 1) A1, A2, A3<sub>1</sub>, A3<sub>2</sub>, A4, A5    2) A1, A2, A3, A3, A5, A4

For instance, while A1 is executing, A2 starts executing. As soon as A1 finishes, A3 starts execution. If A2 finishes execution while A3 is still running, then both *JoinNodes* have one token present. As a result of the non-deterministic behavior of the *DecisionNode*, both A4 and A5 are able to accept the token produced by the first execution of A3. Hence, instead of executing A4 as in the original activity, A5 might be executed. In this case, the second execution of A3 results in the execution of A4.

Besides, the presented approach is limited to a subset of available elements in activities. The behavior might not be

Fig. 10: Redundant element in an *InterruptibleRegion*

preserved in case other elements of activities are used (e.g., guards). In Fig. 10a, an activity is displayed where one instance of the redundant element is part of an *InterruptibleRegion*. As soon as the execution of A3 ends, the execution of A6 (A6 is a substitution for multiple elements in the *InterruptibleRegion*) is also ended. If the transformation is applied and the remaining element stays part of the *InterruptibleRegion*, A6 is always terminated no matter, which predecessor was executed before A3. If the remaining element is no longer part of the *InterruptibleRegion*, A6 is no longer terminated by the execution of A3 as a successor of A1. A possible way to resolve this, is shown in Fig. 10b. By introducing a dummy *ExecutableNode* D as a successor to A3 before A4 and putting the dummy node in the *InterruptibleRegion*, it is still possible to maintain the original behavior. Since this transformation involves an additional *ExecutableNode*, we do not consider this a part of our proposed transformation. Aside from *InterruptibleRegions*, there might be other constellations of elements in activities, where the approach does not preserve the behavior either.

## IX. CONCLUSION AND FUTURE WORK

By investigating a set of UML2 activity diagrams from an industry partner, we showed, that there are activities in practical use, that contain a number of redundant elements. To improve the use of these activities for automated approaches, we proposed a transformation that removes redundant elements while preserving their behavior. The transformation and its property of behavior preservation are based on the assumption of Petri net based semantics. The transformation merges the redundant elements to a single element, adds *ControlNodes*, and connects them to existing elements to assure the preservation of behavior. The number of added *ControlNodes* is in a linear relation to the number of redundant elements. There are special cases that need less *ControlNodes* for the preservation. In order to show the preservation of behavior after transforming the activities, we presented how to derive reachability graphs from an activity and how to compare reachability graphs of different activities. The comparison showed that the transformation preserves the behavior of an activity containing multiple redundant elements with multiple instances. Since the transformation creates a single-entry single-exit structure, we argue that the preservation is valid in general.

Although we argue for the preservation of the behavior, a formal proof for correctness is still needed. There are a number of other formal semantics proposed for UML2 activities. Whether or not all aspects of the preservation hold for these semantics is worth investigating as well as considering all possible constellations of elements in activities.

## REFERENCES

- [1] L. Apfelbaum and J. Doyle, "Model Based Testing," in *Software Quality Week Conference*, 1997.
- [2] J. Hutchinson, M. Rouncefield, and J. Whittle, "Model-Driven Engineering Practices in Industry," in *33rd International Conference on Software Engineering (ICSE)*, 2011.
- [3] Object Management Group (OMG), "OMG Unified Modeling Language (OMG UML), Version 2.5," OMG Document Number formal/2015-03-01 (<http://www.omg.org/spec/UML/2.5/>), 2015.
- [4] E. Sikora, B. Tenbergen, and K. Pohl, "Industry needs and research directions in requirements engineering for embedded systems," *Requirements Engineering*, vol. 17, no. 1, 2012.
- [5] M. Brambilla, J. Cabot, and M. Wimmer, "Model-Driven Software Engineering in Practice," *Synthesis Lectures on Software Engineering*, 2012.
- [6] D. Drusinsky, "From UML activity diagrams to specification requirements," in *IEEE International Conference on System of Systems Engineering (SoSE)*, 2008.
- [7] H. Störrle, "On the Impact of Layout Quality to Understanding UML Diagrams," in *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*. IEEE, 2011.
- [8] W. Liu, S. Easterbrook, and J. Mylopoulos, "Rule-Based Detection of Inconsistency in UML Models," in *Workshop on Consistency Problems in UML-Based Software Development*, vol. 5, 2002.
- [9] D. Firesmith, "Generating Complete, Unambiguous, and Verifiable Requirements from Stories, Scenarios, and Use Cases," *Journal of Object Technology*, vol. 3, no. 10, 2004.
- [10] M. Beckmann and A. Vogelsang, "What is a Good Textual Representation of Activity Diagrams in Requirements Documents?" in *7th International Model-Driven Requirements Engineering Workshop (MoDRE)*, 2017.
- [11] H. Kim, S. Kang, J. Baik, and I. Ko, "Test Cases Generation from UML Activity Diagrams," in *8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD)*, vol. 3, 2007.
- [12] R. Eshuis and R. Wieringa, "Tool Support for Verifying UML Activity Diagrams," *IEEE Transactions on Software Engineering*, vol. 30, no. 7, 2004.
- [13] H. Störrle, "Towards clone detection in UML domain models," *Software & Systems Modeling*, vol. 12, no. 2, 2013.
- [14] M. Beckmann, A. Vogelsang, and C. Reuter, "A Case Study on a Specification Approach using Activity Diagrams in Requirements Documents," in *25th IEEE International Requirements Engineering Conference*, 2017.
- [15] M. Fowler and K. Beck, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.
- [16] T. Mens, "On the Use of Graph Transformations for Model Refactoring," in *Generative and transformational techniques in software engineering*. Springer, 2006.
- [17] T. Mens, G. Taentzer, and D. Müller, "Model-Driven Software Refactoring," *Model-Driven Software Development: Integrating Quality Assurance*, 2008.
- [18] G. Sunyé, D. Pollet, Y. Le Traon, and J.-M. Jézéquel, "Refactoring UML Models," in *International Conference on the Unified Modeling Language*. Springer, 2001.
- [19] A. Correa and C. Werner, "Applying Refactoring Techniques to UML/OCL Models," in *International Conference on the Unified Modeling Language*. Springer, 2004.
- [20] M. Van Kempen, M. Chaudron, D. Kouric, and A. Boake, "Towards Proving Preservation of Behaviour of Refactoring of UML Models," in *Research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*. South African Institute for Computer Scientists and Information Technologists, 2005.
- [21] M. Boger, T. Sturm, and P. Fragemann, "Refactoring Browser for UML," in *International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World*. Springer, 2002.
- [22] M. Misbhauddin and M. Alshayeb, "UML model refactoring: a systematic literature review," *Empirical Software Engineering*, vol. 20, no. 1, 2015.
- [23] K. Altmanninger, "Models in Conflict-Towards a Semantically Enhanced Version Control System for Models," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2007.
- [24] B. Kaur and E. H. Kaur, "Clone Detection in UML Sequence Diagrams Using Token Based Approach," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 5, no. 5, 2015.
- [25] M. Uzam, Z. Li, and M. Zhou, "Identification and elimination of redundant control places in Petri net based liveness enforcing supervisors of FMS," *The International Journal of Advanced Manufacturing Technology*, vol. 35, no. 1, 2007.
- [26] J. Nicolás and A. Tóval, "On the Generation of Requirements Specifications from Software Engineering Models: A Systematic Literature Review," *Information and Software Technology*, vol. 51, no. 9, 2009.
- [27] M. Usman and A. Nadeem, "Automatic Generation of Java Code from UML Diagrams using UJECTOR," *International Journal of Software Engineering and Its Applications*, vol. 3, no. 2, 2009.
- [28] D. Kundu and D. Samanta, "A Novel Approach to Generate Test Cases from UML Activity Diagrams," *Journal of Object Technology*, vol. 8, no. 3, 2009.
- [29] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, 1989.

## INFORMATION EXTRACTION FROM HIGH-LEVEL ACTIVITY DIAGRAMS TO SUPPORT DEVELOPMENT TASKS

---

Published in *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development - Volume 1* (p. 438 - 445) [21].

### TERMS OF USE

*German copyright applies. A non-exclusive, non-transferable and limited right to use is granted. This document is intended solely for personal, non-commercial use.*

### BROADER CONTEXT WITHIN THE THESIS

Chapter 3 and Chapter 4 identified the need to automatically generate textual representations from an activity diagram. This chapter presents a technique that automatically extracts the information necessary to create the textual representation (for possible textual representations and an evaluation of them, please see Appendix B). An activity diagram without redundant elements is assumed as input, which is rendered by the transformation presented in Chapter 5. In the context of this thesis this chapter contributes to the constructive portion and addresses **Problem Statement 2**. This chapter represents contribution V of this thesis. An overview of a prototypical implementation of the presented technique in this chapter can be found in Appendix C. As explained in Subsection 1.4.7, the technique developed in this chapter can also be used beyond the context of the problem statements of this thesis (contribution VI). In order to keep the technique as simple as possible, formal semantics of activities with fewer capabilities than the semantics in Chapter 5 are assumed.

### AUTHOR CONTRIBUTIONS

The author list includes Martin Beckmann, Thomas Karbe, and Andreas Vogelsang. The author of this thesis was the lead author of the publication. He wrote the majority of the article and conceived and developed the technique. Thomas Karbe provided support regarding the formalizations of the concepts and Andreas Vogelsang provided

advice on the research design. Andreas Vogelsang and Thomas Karbe both contributed to the writing of the paper.



# Information Extraction from High-Level Activity Diagrams to Support Development Tasks

Martin Beckmann<sup>1</sup>, Thomas Karbe and Andreas Vogelsang<sup>1</sup>

<sup>1</sup>*Technische Universität Berlin, Ernst-Reuter-Platz 7, 10587 Berlin, Germany  
{martin.beckmann, andreas.vogelsang}@tu-berlin.de, ThomasKarbe@gmx.de*

**Keywords:** UML2 Activity Diagrams, Information Extraction, Activity Semantics

**Abstract:** As the complexity of systems continues to increase, the use of model-driven development approaches becomes more widely applied. One of our industry partners (Daimler AG) uses UML activity diagrams as the first step in the development of vehicle functions, mainly for the purpose of communication and overview. However, the contained information is also valuable for further development tasks. In this paper, we present an automated approach to extract information from these high-level activities. We put a focus on aspects of activities such as propositional logic relations, sequences of actions, and differentiability of execution paths. The extracted parts are needed for the compilation of requirements and the creation of test cases. Also, this approach supports stakeholders unfamiliar with the notations of activities as implicit information is made explicit and hence more accessible. For this purpose, we provide a formalism for the kind of activities our industry partner uses. Based on that formalism, we define properties that express the contained sequences and execution paths. Furthermore, the formalism is used to derive the underlying propositional logic relations. We show how the approach is applied to eliminate hundreds of existing quality issues in an existing requirements document.

## 1 INTRODUCTION

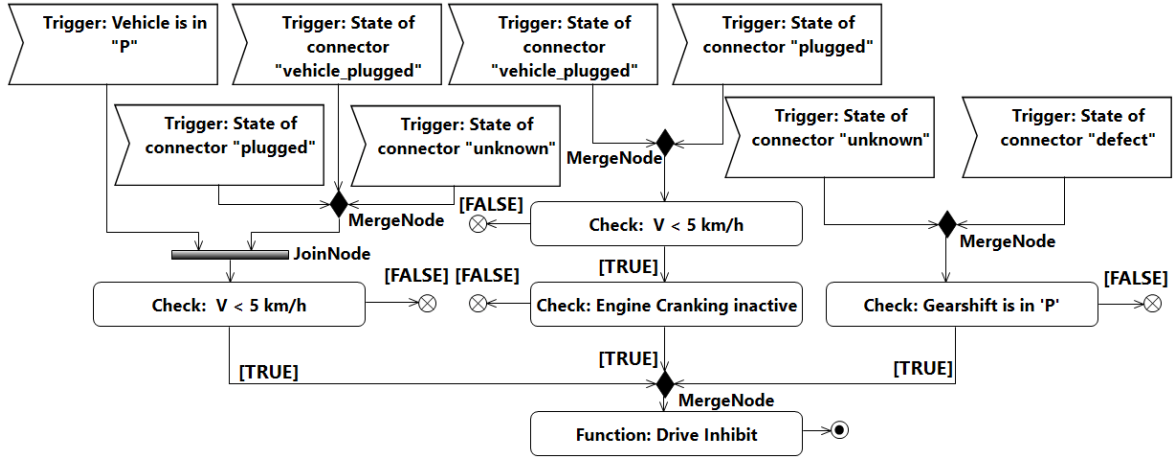
Complex software systems, which, for example, can be found in distributed embedded systems, require model-based and system-oriented development approaches (Broy, 2006). Also, using graphical models for specification manages complexity and improves reusability and analytical capabilities (Vogelsang et al., 2014). One of our industry partners (Daimler AG) uses UML activity diagrams as a first step for developing a new function of a vehicle system. The activities describe the function's activation and deactivation in terms of triggers and conditions that need to be checked and fulfilled before a function is activated. By this, the activity diagrams provide an early overview of the desired function behavior.

Although the main purpose of the diagrams is to be a means of communication and to ease the overall understanding, the contained information is also a valuable input for following development tasks such as the elaboration and documentation of detailed requirements (Drusinsky, 2008) or the derivation of test cases (Kundu and Samanta, 2009). Yet, different tasks have different information needs and may benefit from making explicit specific information contained in the activity diagrams. We aim at support-

ing downstream development tasks by extracting and preparing the relevant information from the activity diagrams. This extraction is additionally helpful for stakeholders unfamiliar with the notations of activity diagrams (Arlow and Neustadt, 2004) because it makes information contained in the activity diagrams more accessible (Maiden et al., 2005).

In this paper, we focus (1) on the transformation of activity diagrams to textual specifications by exploiting information on logical activation expressions and (2) on supporting the derivation of test cases by exploiting information on minimal execution sequences. This paper makes the following contributions:

- We define a simplified representation of UML activities based on graphs. For this simplified representation, we define an algorithm that computes minimal execution sequences within the activity and a second algorithm that computes an activation expression for a function.
- We use the information about minimal execution sequences to derive test cases from the activities.
- We show how we use the activation expressions to derive textual requirements specifications from the activities.
- For both applications, we report on our experiences gained at our industrial partner.

Figure 1: Activity diagram of the function *Drive Inhibit*

## 2 BACKGROUND

Our industry partner uses UML2 activity diagrams to specify functions of a system. These activity diagrams are the first step of the development of a new system function. They are used to get an early overview of the desired function behavior. Although the main focus of the activity is to be a means of communication and to make the understanding easier, it already contains a number of information that can be used in the following development phases such as the elicitation and documentation of requirements and the derivation of test cases.

Figure 1 shows the activity diagram of the function *Drive Inhibit*. The actual behavior of the activated function is described in the *Action* node labeled with *Drive Inhibit* (bottom of the diagram). The function's activation is described by a combination of triggers and checks for conditions. For triggers, the *AcceptEventAction* element is used. The checks are modeled as *Action* elements. If the condition of a check is not fulfilled, the flow ends (*FlowFinal*). As a consequence, a check acts as an implicit AND. The triggers and checks are connected by *Control-Nodes* such as *JoinNodes* and *MergeNodes*. *JoinNodes* act as synchronization points and can be interpreted as AND operators in terms of propositional logic. *MergeNodes* represent OR operators. Once the actual functionality of the function is executed, *ActivityFinal* elements designate the end of an activity.

Relevant information for our industry partner concerns (amongst others): (minimal) execution paths, propositional logic relations and sequential or independent executability of actions.

Execution paths are of interest for testing and to facilitate the planning of the system. They are the basis to derive test cases that ensure that the function

is in fact activated, when certain action are executed. The execution paths also provide information about the sequences of execution of *actions*. This can be combined with the mapping of the involved actions to the components of the system (this mapping is not part of the activity). As a result, it is possible to make statements on the dependencies between the involved components. This knowledge is applied during the planning of the development of the system.

Propositional logic relations are needed to derive requirements that describe the correct behavior of the system as well as the test cases that validate these requirements.

## 3 RELATED WORK

As this paper introduces a formalism for a certain kind of activities, it is related to work about formal semantics of UML2 activities. The UML2 Specification describes *Activities* as Petri net like graphs (Object Management Group (OMG), 2015, p. 283), but does not provide formal semantics. Therefore a number of formal semantics have been proposed, i.a. (Störrele, 2004). While most approaches try to cover the capabilities to a full extent, it is considered useful to express activities in simpler constructs (Lano, 2009). We use this idea and present a formalism solely devoted to derive information about certain aspects in activities.

Graphical models are the basis for a number of approaches that derive different software engineering artifacts from the models. Amongst others, they are used to automatically generate source code (Usman and Nadeem, 2009) and test cases (Kundu and Samanta, 2009). Using graphical models and espe-

cially UML to generate textual requirements or parts of requirements documents has already been covered by a number of research papers (Nicolás and Toval, 2009). Specifically activities as a source for requirements have already been addressed by Drusinsky (Drusinsky, 2008), however, only for UML-1. Additionally, we take into account propositional logic relations, execution paths, and allow for queries on actions about independent executions.

In contrast to the mentioned approaches, our approach focuses on extracting certain aspects of activities and does not restrict itself on a single application.

## 4 EXTRACTING INFORMATION FROM ACTIVITIES

For the purpose of this paper, we aim at extracting specific information from activities to facilitate downstream development tasks. More specifically, we want to extract the following information:

**Independent Actions.** Independent actions within an activity can be executed without any interrelations. This information is useful for the planning of the development. Actions are executed by components of the system. From the independence of actions follows that there is no flow of information between the components and hence development can progress without considering the component executing an independent action.

**Minimal Execution Paths.** A minimal execution path for a node within an activity is a set of actions that need to be executed before the node can be executed. These paths contain all actions that are logically required for a token to reach an action. Superfluous actions occurring in parallel are not part of the minimal execution path. This information is useful for the creation of test cases. The test cases verify that a function is executed due to or in spite of certain conditions. Using minimal paths ensures that only conditions are tested that influence the examined executed path. This leads to a minimal set of tests, which are necessary to confirm the behavior of a function.

**Activation Expressions.** An activation expression for a node within an activity is a propositional expression that reflects the logical relations between the preceding actions of the node. The activation expression abstracts from any order of execution and can be used to derive textual specifications corresponding to the activities.

In the following, we present how these information can be extracted from the activities.

### 4.1 Activity Graphs

To extract the information on independent actions and minimal execution paths, we introduce activity graphs as a simplified representation of the activities. Activity graphs focus on expressing whether certain actions are independent of one another or whether they have to be executed sequentially. We transform an activity to an activity graph by mapping the actions of an activity to nodes of a graph. We assume that implicit connections in the activity are made explicit and that *ExecutableNodes* only appear once in the activity. Beckmann et al. have proposed an approach that we use to remove redundant occurrences of *ExecutableNodes* within an activity (Beckmann et al., 2017a). There may be cycles in the activity.

Each node in the activity graph has a label containing the text of the corresponding *Action* of the activity. They also have one of the following types: Trigger, Check, Function, Merge, Decision, Join, Fork. Moreover, each node has a set of successors.

#### Definition 1. Activity Graph

Given a non-empty set of nodes  $V$ , an activity graph  $T$  is defined as

$$T \stackrel{\text{def}}{=} (V, \text{succ}_T, \text{type}_T, \text{label}_T)$$

where

1.  $\text{succ}_T : V \rightarrow \mathcal{P}(V)$  is the successor function for  $T$ , where  $\text{succ}_T(v)$  denotes the set of all successor nodes of  $v \in V$ ,
2.  $\text{type}_T : V \rightarrow \{\text{Trigger}, \text{Check}, \text{Function}, \text{Merge}, \text{Decision}, \text{Join}, \text{Fork}, \text{End}\}$  assigns a type to every node, and
3.  $\text{label}_T : V \rightarrow \Sigma^*$  assigns a label to every node.

#### Definition 2. Direct Predecessors

Given an activity graph  $T$ , the set of direct predecessors of a node  $v \in V$  is defined as

$$\text{dpred}_T(v) \stackrel{\text{def}}{=} \{w \mid v \in \text{succ}_T(w)\}$$

In the activity the direct predecessor is the source node of any incoming edge. There might be more than one direct predecessor to one node. Since we assume that all connections were made explicit and there are no redundant elements, multiple direct predecessors occur only for *JoinNodes* and *MergeNodes*.

#### Definition 3. Execution Sequence

Given an activity graph  $T$ ,

1. A list of nodes  $s = \langle v_1, \dots, v_n \rangle$  with  $v_1, \dots, v_n \in V$  is called an execution sequence, and  $v_i$  is called the  $i$ -th execution step of  $s$ .
2. An execution step  $v_i$  is a sequence-predecessor of another execution step  $v_j$  (denoted by  $v_i <_s v_j$ ) if  $i < j$ .

3. The set of all execution sequences of  $T$  is denoted by  $S$ .

Considering Figure 1 one possible execution sequence might be *Trigger: State of connector "unknown", Check: Gearshift is in 'P', Function: Drive Inhibit*.

**Definition 4. Prefix**

Given an activity graph  $T$  and an execution sequence  $s = \langle v_1, \dots, v_n \rangle$ . For any  $k$  with  $1 \leq k \leq n$  the  $k$ -prefix (or just prefix) of  $s$  is defined by

$$s_{(k)} \stackrel{\text{def}}{=} \langle v_1, \dots, v_k \rangle.$$

**Definition 5. Node Count**

Given an activity graph  $T$  and an execution sequence  $s = \langle v_1, \dots, v_n \rangle$ . For any node  $v \in V$ , the node count of  $v$  in  $s$  is a function  $\#_v(s) : S \rightarrow \mathbb{N}$  and describes the number of appearances of  $v$  in  $s$ .

Example:

- $\#_a(\langle a, b, c, a, d, e, a, d \rangle) = 3$ ,
- $\#_e(\langle a, b, c, a, d, e, a, d \rangle) = 1$ ,
- $\#_f(\langle a, b, c, a, d, e, a, d \rangle) = 0$ .

**Definition 6. Valid Execution Sequence**

Given an activity graph  $T$  and an execution sequence  $s = \langle v_1, \dots, v_n \rangle$ ,

1. An execution step  $v_i$  of a sequence  $s$  is valid (denoted by  $s \vdash_T v_i$ ) if and only if one of the following cases is true:

- (a)  $d\text{pred}_T(v_i) = \emptyset$ ,
- (b)  $d\text{pred}_T(v_i) \neq \emptyset \wedge \text{type}(v_i) \neq \text{Merge} \wedge \forall w \in d\text{pred}_T(v_i). \#_w(s_{(v_i)}) \geq \#_{v_i}(s_{(v_i)})$   
*Explanation: A join node is valid when each of its predecessors appears at least as often as the join node itself in the prefix before it. Check, Function, Fork, Decision, and End nodes are similar, but have only one predecessor. The formula is the same for them.*

- (c)  $d\text{pred}_T(v_i) \neq \emptyset \wedge \text{type}(v_i) = \text{Merge} \wedge \#_{v_i}(s_{(v_i)}) \leq \sum_{w \in d\text{pred}_T(v_i)} \#_w(s_{(v_i)})$   
*Explanation: A merge node is valid when all its predecessors together appear at least as often as the merge node itself in the prefix before it.*

2. An execution sequence  $s$  is valid (denoted by  $\vdash_T s$ ), when all its execution steps are valid.
3. The set of all valid execution sequences for  $T$  is denoted by  $S_T$ .

**Definition 7. Predecessor**

Given an activity graph  $T$ , a node  $v_i \in V$  is a predecessor of another node  $v_j \in V$  (denoted by  $v_i <_T v_j$ ) if  $v_i$  is a sequence predecessor of  $v_j$  in every valid sequence of  $T$ .

$$v_i <_T v_j \Leftrightarrow \forall s \in S_T. v_i <_s v_j$$

This definition is used to find dependencies between actions. In case a node is predecessor of another node, the predecessor has to be executed first. This also tells us that there is an interaction between nodes.

**Definition 8. Independent Nodes / Parallel Executable**

Given an activity graph  $T$ , two nodes  $v_i, v_j \in V$  are independent (denoted by  $v_i \parallel_T v_j$ ) if they are not predecessors of each other:

$$v_i \parallel_T v_j \Leftrightarrow v_i \not<_T v_j \wedge v_j \not<_T v_i$$

In contrast to the predecessor relation, two independent nodes can be executed without any interrelations between the involved actions. An example in Figure 1 are the checks  $V < 5 \text{ km/h}$  and *Gearshift is in 'P'*.

**Definition 9. Minimal Execution**

Given an activity graph  $T$  and a node  $v \in V$ . A minimal execution sequence  $s_{\text{min},T}(v) = \langle v_1, \dots, v_n \rangle$  is a valid execution sequence that ends in  $v$  and for which no  $i$  exists for which  $1 \leq i < n$  and  $\langle v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_n \rangle$  is valid.

Note, that  $v_n = v$  because the sequence ends in  $v$ .

*Explanation: An execution sequence is minimal when no step can be cut out of the sequence.*

Every path to the specified node that does not contain unnecessary actions for the activation, is a minimal execution. Since *MergeNodes* might have multiple predecessors, there can be more than one minimal execution. The action *Check: V < 5 km/h* after the *JoinNode* in Figure 1 has three minimal executions. Each path consists of one of the three triggers connected by the *MergeNode*, the *MergeNode* itself, the *JoinNode* and the action *Trigger: Vehicle is in 'P'*.

**Definition 10. Concatenation of Execution Sequences**

Given two execution sequences of disjoint nodes  $s_1 = \langle v_1, \dots, v_n \rangle$  and  $s_2 = \langle w_1, \dots, w_m \rangle$ . The concatenated execution sequence  $s_1 \circ s_2$  is defined as

$$s_1 \circ s_2 \stackrel{\text{def}}{=} \langle v_1, \dots, v_n, w_1, \dots, w_m \rangle$$

The algorithm to compute a minimal execution is shown in Algorithm 1. The algorithm works recursively through the graph. In each step the necessary minimal executions are concatenated to the current node. Which executions are necessary depends on the type of the node. In case a node is neither a *JoinNode* nor a *MergeNode*, the minimal execution is the concatenation of the minimal execution of its direct predecessor and itself. For a *JoinNode*, all previous minimal executions are needed. For a *MergeNode*, any of the predecessor can be used. Hence,

**Algorithm 1** Recursively computing a minimal execution

---

**Input:** Activity Graph  $T$ , Node  $v \in V$

**function** MINEX( $v$ )

**if**  $dpred_T(v) = \emptyset$  **then**

**return**  $\{v\}$

**else if**  $type(v_i) \notin \{Merge, Join\}$  **and**

$dpred_T(v) = \{w\}$  **then**

**return** MINEX( $w$ )  $\circ \langle v \rangle$

**else if**  $type_T(v) = Join$  **and**

$dpred_T(v) = \{w_1, \dots, w_n\}$  **then**

**return** MINEX( $w_1$ )  $\circ \dots \circ$  MINEX( $w_n$ )  $\circ \langle v \rangle$

Note, this step is not deterministic, since depending on the order of concatenation there are multiple options. Only one choice is needed.

**else if**  $type_T(v) = Merge$  **and**

$w \in dpred_T(v)$  (any predecessor) **then**

**return** MINEX( $w$ )  $\circ \langle v \rangle$

Note that this step is not deterministic, since multiple predecessors can exist. Any choice would be correct.

**end if**

**end function**

---

there are multiple minimal executions. The algorithm terminates, if there are no predecessors or if a cycle is detected. Executions containing cycles are discarded, because they cannot be minimal executions.

## 4.2 Activation Expressions

Actions that are predecessors of other actions in an activity diagram can also be interpreted as logical facts that need to be fulfilled before an action can be executed. Activation expressions focus on these logical relations between actions. These relations can be represented by a propositional logic expression tree. The algorithm to construct the activation expression for a node in an activity graph is displayed in Algorithm 2.

The algorithm requires the node for which the activation expression shall be computed as input. In our case, we are especially interested in action nodes that represent function executions. Some of the activities of our industry partner contain more than one function. In that case, multiple trees have to be created since each function has different triggers and checks, and thus, the activation expression is also different. As a second input, the algorithm requires a node of the tree that is to be created. The input is required since the algorithm works recursively. When the al-

**Algorithm 2** Recursively computing an expression tree

---

**Input:** Node  $v_{Act} \in V_{Act}$ , Node  $v_{Tree} \in V_{Tree}$

**function** CREATEEXPTREE( $v_{Act}, v_{Tree}$ )

**if**  $dpred_{Act}(v_{Act}) = \emptyset$  **then**

$sucTree(v_{Tree}) = sucTree(v_{Tree}) \cup v_{Act}$

**else if**  $dpred_{Act}(v_{Act}) \neq \emptyset$  **and**  $type_{Act}(v_{Act}) \in \{Trigger, Check, Function\}$  **then**

$v_{Treenext} \stackrel{\text{def}}{=} createNode(AND)$

$sucTree(v_{Tree}) = sucTree(v_{Tree}) \cup v_{Treenext}$

$sucTree(v_{Treenext}) = sucTree(v_{Treenext}) \cup v_{Act}$

$v_{Act} \stackrel{\text{def}}{=} v \in dpred_{Act}(v_{Act})$

$v_{Tree} \stackrel{\text{def}}{=} v_{Treenext}$

        createExpTree( $v_{Act}, v_{Tree}$ )

**else if**  $dpred_{Act}(v_{Act}) \neq \emptyset$  **and**

$type_{Act}(v_{Act}) = Join$  **then**

$v_{Treeand} \stackrel{\text{def}}{=} createNode(AND)$

$sucTree(v_{Tree}) = sucTree(v_{Tree}) \cup v_{Treeand}$

**for all**  $v_{Actin}$  **of**  $dpred_{Act}(v_{Act})$  **do**

$v_{Act} \stackrel{\text{def}}{=} v_{Actin}$

$v_{Tree} \stackrel{\text{def}}{=} v_{Treeand}$

            createExpTree( $v_{Act}, v_{Tree}$ )

**end for**

**else if**  $dpred_{Act}(v_{Act}) \neq \emptyset$  **and**

$type_{Act}(v_{Act}) = Merge$  **then**

$v_{Treeor} \stackrel{\text{def}}{=} createNode(OR)$

$sucTree(v_{Tree}) = sucTree(v_{Tree}) \cup v_{Treeor}$

**for all**  $v_{Actin}$  **of**  $dpred_{Act}(v_{Act})$  **do**

$v_{Act} \stackrel{\text{def}}{=} v_{Actin}$

$v_{Tree} \stackrel{\text{def}}{=} v_{Treeor}$

            createExpTree( $v_{Act}, v_{Tree}$ )

**end for**

**else if**  $dpred_{Act}(v_{Act}) \neq \emptyset$  **and**

$type_{Act}(v_{Act}) \in \{Fork, Decision\}$  **then**

$v_{Act} \stackrel{\text{def}}{=} v \in dpred_{Act}(v_{Act})$

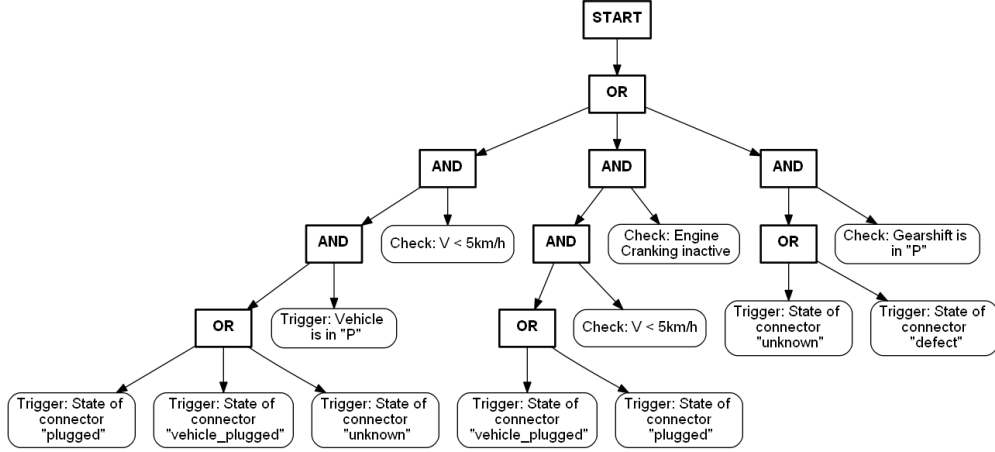
        createExpTree( $v_{Act}, v_{Tree}$ )

**end if**

**end function**

---

gorithm is called for the first time, a *start* node is used as the root node of the tree. What the algorithm basically does, is to traverse the activity graph backwards. It starts from the node that represents the function that has to be activated. From there the predecessors are analyzed until the triggers of the function or nodes without any predecessors are reached. As a consequence, the algorithm terminates as long as there is

Figure 2: Expression Tree of the function *Drive Inhibit*

no cycle in any of the execution sequences. This can be automatically ensured beforehand by checking for cycles. Also, for the activities our industry partner uses, a detected cycle can be ignored. This is possible, since the repeated execution of actions does not have any influence on the function activation. If the actions in the cycle were executed once, the flow of tokens also continues outside the cycle. Further repetitions do not effect that flow.

In each step of the traversal, the type of the current activity node is examined. Depending on the type, the nodes that are appended to the tree, differ. In case the examined node is an *Action* (e.g., a check), it means it has to be executed successfully for the flow to continue. This is depicted in Figure 3a. The traversal of the activity starts from the function. Before the function can be executed, a check must be fulfilled. Besides, there might be other nodes before the check. As this represents an *AND* connection, an *AND* node is added to the tree, and the found check is added to that new *AND* node. The resulting tree is shown in Figure 4a. The following recursive call uses the added *AND* node as the tree node input. The following activity nodes are then added to the *AND*. If a *JoinNode* or *MergeNode* is found in the activity, an *AND* or *OR* node is appended to the tree respectively. In contrast to a single action, these *ControlNodes* might have more than one predecessor. Exemplar activities for the *JoinNode* and the *MergeNode* are shown in Figure 3b and Figure 3c respectively. All predecessors are added to these tree nodes. The corresponding expression trees to the activities in Figure 3b and Figure 3c are shown in Figure 4b and Figure 4c. There is no negation operator, since there are no actions that undo events and hence stop the flow of tokens.

The corresponding expression tree to the activity in Figure 1 is displayed in Figure 2. The tree nodes

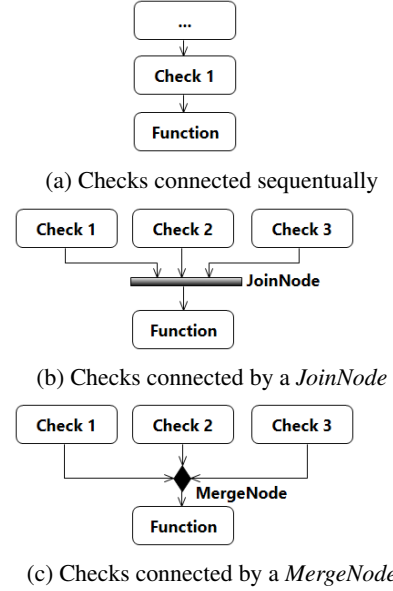


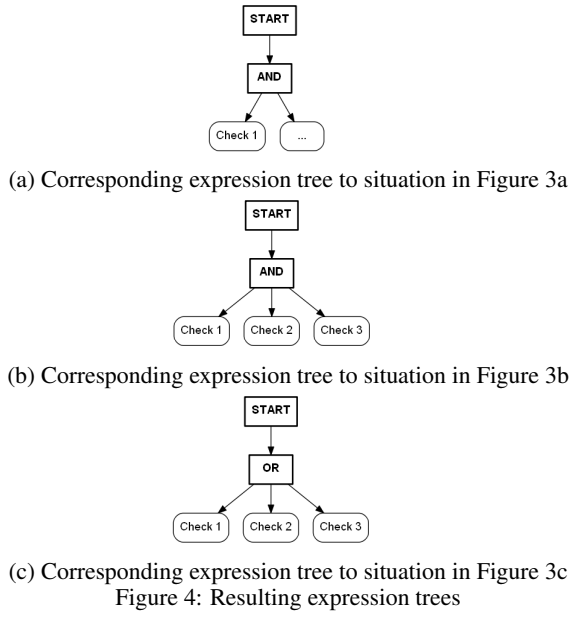
Figure 3: Different situations in activities

that represent the operators (START, AND, OR) are displayed in square boxes, while the actual *ActivityNodes* are displayed as oval boxes. As a result of the algorithm, the *ExecutableNodes* of the original activity are the leaves of the tree.

## 5 APPLICATIONS AND LIMITATIONS

### 5.1 Applications

We used the introduced algorithms and definitions to support different development tasks in practice.



### 5.1.1 Transformation of Activity Diagrams to Textual Specifications

In industry, graphical models such as activity diagrams cannot be used as the sole means of specification. Textual requirements complementing the activities are needed because of legal considerations (Sikora et al., 2012; Maiden et al., 2005) and to provide a systematic display of derived information (Weber and Weisbrod, 2002). Recent studies have found that practitioners prefer textual requirements specifications that are structured according to the different logical cases that may lead to a specific event (Beckmann and Vogelsang, 2017). Therefore, we used the structure of the activation expression tree to generate complementing textual requirements specifications for 36 activity diagrams of our industry partner. That way, we eliminated hundreds of different existing quality issues of a previous version.

Figure 5 shows the textual requirements derived from the activity of Figure 1. The excerpt shows explicitly the propositional logic relations by using the operators *AND* and *OR*. All elements connected by the same operator were placed one level below. This kind of structure equals the structure of the activation expression tree. Hence, we could directly map the result of the underlying propositional logic to the document structure. Prior studies have shown that manual creation and maintenance of textual requirements from diagrams is error-prone and labor-intensive (Beckmann et al., 2017b). An automatic model-to-text transformation based on our algorithm prevents quality issues and may save time.

Text	Level	Type
<b>Drive Inhibit</b>	2	Function
OR	3	-
AND	4	-
Vehicle is in "P"	5	Trigger
OR	5	-
State of connector "plugged"	6	Trigger
State of connector "vehicle_plugged"	6	Trigger
State of connector "unknown"	6	Trigger
V < 5 km/h	5	Check
AND	4	-
OR	5	-
State of connector "plugged"	6	Trigger
State of connector "vehicle_plugged"	6	Trigger
V < 5 km/h	5	Check
Engine Cranking inactive	5	Check
AND	4	-
OR	5	-
State of connector "defect"	6	Trigger
State of connector "unknown"	6	Trigger
Gearshift is in "P"	5	Check

Figure 5: Derived Textual Requirements

### 5.1.2 Derivation of Test Cases

The approach was applied to recreate parts of already existing test cases for the displayed function *Drive Inhibit* in an automatic manner as a proof-of-concept. These parts encompass the name of the test case as well as templates for the test steps that must be performed to conduct the test case. The test steps must be added manually as they are not part of the activity diagram. The test cases ensure that the function is activated due to certain occurring events and fulfilled conditions. The necessary states and circumstances were directly derived from the identified minimal executions. The minimal executions of an action in the activity contain all necessary actions (i.e., events) that must appear and conditions that must be fulfilled to start an execution. As a result, test cases that describe in which states a function is activated can be directly derived since a minimal execution only contains these necessary conditions. Consequently every minimal execution is used to derive one test case. The created test cases can therefore assure that the function is in fact executed under the intended circumstances. Hence, using this approach ensures that all necessary conditions for executions are tested. For example in Figure 1 this leads to the creation of seven test cases. Three test cases originate from the three triggers connected to the trigger *Vehicle is in "P"* by a *JoinNode*. Two test cases are created for each pair of the two triggers connected by the *MergeNodes*.

In addition, non-minimal sequences can also be useful. The execution of superfluous actions makes sure the function is still activated when the necessary actions were executed. Also, it can be checked whether the function is activated, although necessary conditions are not met. In that case necessary actions are not executed.

## 5.2 Limitations

We focused on the capability of activities to describe sequences, parallelism, execution paths and propositional logic relations. Still, activities can be used in other ways to describe other aspects of behavior. Consequently, it is not possible to foresee every application. Thus, it is necessary to restrict oneself to certain aspects. While the extracted information can be used for multiple purposes, there are use cases that require different aspects our approach does not yet cover. One of these aspects are asynchronous events that are potentially used to abort the execution of an activity. These were not part of our work, since our industry partner does not use them.

Also, this work focuses on the activity diagrams of our industry partner. These activities only incorporate a subset of elements in activities. Still, this kind of description is quite common to describe functions (Firesmith, 2004). As a result, the approach is not generally applicable but we think that it provides a benefit for that kind of graphical descriptions.

## 6 CONCLUSION AND OUTLOOK

In this paper we presented an approach to extract implicitly contained information from high-level activities to support downstream development tasks. For this purpose we introduce activity graphs as a simplified, yet formal, representation of activity diagrams, which can be used to make statements about sequences and execution paths of activities. We show in detail how this can be used to derive textual requirements, which both improves the quality of the resulting requirements document and saves effort in its creation. Also, the creation of test cases was performed as a proof-of-concept for one function.

Furthermore, it is planned to use the extracted information for impact analysis. By combining the activities with the mapping of the actions to the components, dependencies between components are made more easily accessible. This knowledge will be used to derive visual architectural views of the whole system, which in turn shall facilitate release planning.

As the approach is restricted to a subset of elements, the approach is not generally applicable to all activities. Incorporating all elements (such as guards) of activities into the approach is an open issue. Also, there are further aspects of activities that are needed during the development of systems we did not yet consider. Which aspects need to be included and what artifacts they might be used for is also worth investigating.

## REFERENCES

- Arlow, J. and Neustadt, I. (2004). *Enterprise patterns and MDA: Building better software with archetype patterns and UML*. Addison-Wesley Professional.
- Beckmann, M., Michalke, V., Vogelsang, A., and Schlutter, A. (2017a). Removal of Redundant Elements within UML Activity Diagrams. In *Conference on Model Driven Engineering Languages and Systems*.
- Beckmann, M. and Vogelsang, A. (2017). What is a Good Textual Representation of Activity Diagrams in Requirements Documents? In *Model-Driven Requirements Engineering Workshop*.
- Beckmann, M., Vogelsang, A., and Reuter, C. (2017b). A Case Study on a Specification Approach using Activity Diagrams in Requirements Documents. In *International Requirements Engineering Conference*.
- Broy, M. (2006). Challenges in automotive software engineering. In *Proceedings of the 28th international conference on Software engineering*.
- Drusinsky, D. (2008). From UML activity diagrams to specification requirements. In *International Conference on System of Systems Engineering*.
- Firesmith, D. (2004). Generating Complete, Unambiguous, and Verifiable Requirements from Stories, Scenarios, and Use Cases. *Journal of Object Technology*.
- Kundu, D. and Samanta, D. (2009). A Novel Approach to Generate Test Cases from UML Activity Diagrams. *Journal of Object Technology*.
- Lano, K. (2009). *UML 2 Semantics and Applications*. John Wiley & Sons.
- Maiden, N. A., Manning, S., Jones, S., and Greenwood, J. (2005). Generating requirements from systems models using patterns: a case study. *Requirements Engineering*.
- Nicolás, J. and Toval, A. (2009). On the generation of requirements specifications from software engineering models: A systematic literature review. *Information and Software Technology*.
- Object Management Group (OMG) (2015). *OMG Unified Modeling Language (OMG UML), Version 2.5*.
- Sikora, E., Tenbergen, B., and Pohl, K. (2012). Industry needs and research directions in requirements engineering for embedded systems. *Requirements Engineering*.
- Störrle, H. (2004). Semantics of UML 2.0 Activities. In *Symposium on Visual Languages and Human-Centric Computing*.
- Usman, M. and Nadeem, A. (2009). Automatic Generation of Java Code from UML Diagrams using UJECTOR. *Journal of Software Engineering and Its Applications*.
- Vogelsang, A., Eder, S., Hackenberg, G., Junker, M., and Teufl, S. (2014). Supporting concurrent development of requirements and architecture: A model-based approach. In *Conference on Model-Driven Engineering and Software Development*.
- Weber, M. and Weisbrod, J. (2002). Requirements Engineering in Automotive Development - Experiences and Challenges. In *International Conference on Requirements Engineering*.



## CONCLUDING DISCUSSION

---

In this chapter, a summary of the contributions of this thesis is first given (Section 7.1). Next, Section 7.2 discusses how the contributions relate to one another, how the contributions might be applied outside of the examined use case, and further aspects of the contributions from Chapters 3, 4, and 6 that have not yet been discussed in the publications.

### 7.1 SUMMARY

This thesis examines the use of a specification approach that employs coexisting graphical and textual representations of requirements in practice. For this purpose, a specification approach was examined that uses UML2 activity diagrams in combination with hierarchically structured text. Its unique property and focal point of the thesis is the fact that this approach is designed, implemented, and productively used by practitioners themselves. These circumstances are important to consider, since in practice, specification approaches are influenced by organizational and domain specific peculiarities that cannot be fully predicted from a research perspective. This thesis consists of two portions: an analytical portion and a constructive portion. In the analytical portion, we contribute to the knowledge on comparable specification approaches and identify a practical problem and assess its relevance. In the constructive portion, we offer contributions that can be applied to the examined case to address the previously identified problem. Hence, both portions constitute a holistic contribution to the improvement of comparable existing specification approaches or considerations for new ones.

**Analytical Portion.** This portion of the thesis was mainly concerned with gathering information on the examined approach and therefore addresses **Problem Statement 1**. In an attempt to gain deeper insight into the specification approach, we examined how practitioners deal with the specification approach itself as well as with its outcomes in a first study (Chapter 3). As a result, the answer to the first question in Section 2.5, regarding the implications of such a specification approach in practice, has many elements. Generally, the current implementation of the approach is perceived as beneficial. A major advantage is the visual representation of the activity diagrams. This is seen as helpful, especially during the initial steps of development. However, the textual representations are still considered the main specification artifacts. As such, the textual representation serves its own

purposes, i.e., as a vehicle for more detailed information and, because of its legally binding nature as the preferred medium to communicate with suppliers.

Nonetheless, a number of issues complicate the use of such a specification approach. First, the form of the representation is not necessarily catered to every person that uses the content. Also, different tools for different representations complicate the handling of the available information, which leads to inconsistencies, among other issues. These inconsistencies are the focal point of a further study (see Chapter 4). This study confirms that there are numerous types of inconsistencies between the representations, as well as other quality issues. For the purpose of assessing the different types of inconsistencies and other quality issues, categories were defined that represent groups of mutual properties within the deficiencies found. For the examined system, how often the categories appear was counted. In addition, the categories were assessed by the practitioners. The number of appearances in combination with the assessment of the categories demonstrated the negative impact each category has on the work of users. Ultimately, the study yielded conclusive results as the practitioners agreed that especially the inconsistencies have a significant impact on the applicability of the specification approach.

**Constructive Portion.** In the first study of this thesis, we found that inconsistencies between the coexisting representations impair the use of the specification approach. In the second study, we elaborated on this issue and found that the inconsistencies go so far as to constitute a significant impact and hence present a problem of decisive relevance for the approach's applicability in practice. Following the conclusions of the studies, we presume that the circumvention of inconsistencies from the outset is a valid answer to the second question in Section 2.5. As a result, the constructive portion of this thesis is dedicated to addressing the problem defined by **Problem Statement 2**.

More specifically, we developed an approach that generates textual representations from the activity diagrams with the purpose of preventing inconsistencies and other quality issues. Chapter 5 presents an automatable transformation that creates the foundation for the generation by ensuring that each activity only contains unique *ExecutableNodes*. This transformation was shown to preserve the behavior of the activity such that the generated textual representation is not affected. A technique is presented that extracts information from the activities built on the outcome of the transformation. This information can be utilized to construct textual representations that can be tailored to the needs of different readers.

## 7.2 DISCUSSION

First, this section discusses how the individual contributions of this thesis are connected to one another. Then, the applicability of the findings are addressed. Finally, further aspects that have thus far remained unmentioned in this thesis are discussed.

### 7.2.1 *Interrelations of the Contributions*

The main theme of this thesis is the examination of a specification approach in practice. In this context, we have set out to determine whether work in this research area meets expectations when applied in an industrial context. We have also engaged in the development of a solution for a problem that arises in a specific instance of the specification approach analyzed in this thesis. These two elements of this thesis differ in their intentions and scopes. Nevertheless, in order to conduct a systematic research effort, the development of a solution must be preceded by an analysis and an assessment of the problem identified by practitioners. As a result, not only are the contributions of the individual parts of this thesis interwoven, but the analytical portion forms the basis for the actual problem solution. Thus, each contribution influences every subsequent contribution in some way.

Contribution I represents the foundation of all other contributions. Its insights lead to a closer examination of the inconsistencies between the coexisting specification artifacts in contribution II. Contribution II confirms our initial impression of the negative impact of inconsistencies. In terms of design science, both contributions are required as they are part of the scientific search process and because they define the relevance of the problem.

The contributions of the constructive portion build upon these results, as they prove that the problem is worth to be considered. Contribution III is the first step in preventing inconsistencies and other quality issues, as well as other shortcomings that arise from the application of comparable specification approaches. Unfortunately, contribution III does not solve the identified problem. Even if the suggestions are followed, without the support of an automatic approach to ensure consistency, manual task execution makes the appearance of inconsistencies unavoidable. Hence, it is imperative to automate the derivation of the textual representation from the [UML2](#) activities. In this sense, contributions IV & V realize one of the propositions of contribution III. Contribution IV is necessary for enabling the information extraction, which represents contribution V and hence must be considered its prerequisite.

### 7.2.2 *Applicability of the Contributions*

As discussed in Section 2.5, making academic research relevant to practitioners remains an ongoing problem for the academic community [110]. In the context of RE, Gorschek et al. describe their experience of translating their academic work into practice and propose a model to enable technology transfer from research to practice [94].

While we did not follow their model exactly, many of their suggested seven steps were performed as part of our research effort. Their first step demands basing research on industry needs. As our research effort arose from the needs of practitioners, this step was inherently fulfilled. Their second step, which calls for a research agenda, was also addressed, since we conducted the research in a systematic manner (following the guidelines for design science from Hevner et al. [108]). Regarding the research agenda, Gorschek et al. emphasize the importance of researchers acting as a link to the state of the art (research rigor). In Chapter 2, we examined existing related work, meaning this thesis meets the criteria for scientific work. Next, Gorschek et al. suggest creating a candidate solution (step 3) and then validating it in multiple iterations (step 4 - step 6). These iterations increase the involvement of practitioners each time – moving from lab conditions to a real-world pilot. At this point we deviated from their model. The candidate solution was refined in multiple iterations but this was based on feedback from the industry partner throughout the entire process. Their final step relates to the release of the solution. Similar to Bajec et al. [20], Gorschek et al. note that in practice, the individual conditions of different organizations make it necessary to tailor the solution to each organization's needs. As this work was created to meet the needs of a specific industry partner, this requirement for applicability in practice was inherently fulfilled. However, it is open to discussion to what extent the findings can be applied elsewhere. In fact, the applicability in other contexts varies widely for each contribution.

Contribution I is considered the most widely applicable of all the contributions. In comparison to contribution II, contribution I is less focused on the actual use of activity diagrams and the examined textual representation. Nevertheless, the categories of the inconsistencies and other quality issues identified and assessed in contribution II are likely to occur similarly in other implementations of such a specification approach (see also: Subsection 7.2.3).

The situation for contribution III is similar to the situation of contributions II. Contribution III builds upon the results of the preceding contributions (contributions I and II). But since contribution I is more widely applicable than contribution II and inconsistencies are likely to be an issue for comparable specification approaches that use different models and textual representations, it is not unrealistic to assume

that the suggestions offered by this contribution could be applied to different implementations of these types of approaches.

Contribution IV is solely focused on UML2 activities and therefore its use is restricted to a certain model type and is less widely applicable in the context of specification approaches that use coexisting artifacts. Nonetheless, it may be used for UML2 activities utilized in other contexts. Still, its applicability in other contexts is restricted by the semantics of the UML2 activities assumed in this thesis. Beyond this restriction, it seems to be reasonable that the underlying transformation could be applicable to other data flow diagram types (e.g., BPMN).

The scope of contribution V is the most focused of all the contributions of this thesis. It essentially represents a continuation of contribution III and specifically addresses the circumstances of the examined case in this thesis. Similar to contribution III, it builds on the findings of all the preceding contributions. However, contribution V dismisses the abstractions of the prior contributions and focuses on the implementation of a specification approach using the coexisting representations of requirements of this thesis. Considering these conditions, contribution V provides an automatic technique for extracting information from UML2 activity diagrams that follow a predefined pattern. Nevertheless, the technique is not restricted to this specific use case, as it is possible that applications may reach beyond the generation of textual representations of requirements. However, this technique is still limited to a certain type of UML2 activity diagrams.

### 7.2.3 Further Aspects

In this subsection, we discuss additional points related to Chapters 3, 4, and 6. These were not mentioned in the respective articles due to their limited scope, but nonetheless these points are of importance within the overall context of this thesis.

#### 7.2.3.1 Coexisting Graphical and Textual Representations of Requirements

The main intention of Chapter 3 was to determine how coexisting representations of requirements are perceived in practice. Related to the specific context of this thesis, similar research efforts have already been undertaken by the MDD community [222]. As one of the examined representations in this thesis is a model in graphical form, it can be assumed that the findings are strongly related – especially regarding aspects that refer to the model. The results show that the aims MDD pursues have “the potential to greatly reduce development time” and “will help us develop better software systems faster” [36], and are perceived in a similar fashion by the practitioners of the examined case in this thesis. Beyond this, the findings also confirm the argument of works that find the use of models to be primarily more informal. Fol-

lowing Störrle's classification of different types of models, the activity diagrams in this thesis can be considered *informal models* or perhaps *semi-formal models*. Thus, the examined case confirms Störrle's findings about the use of models in industrial practice as being *informal models* or *semi-formal models* most of the time [222]. Hence, the models are largely used in the sense of MBE in a supportive role rather than as primary artifacts, as MDD intends.

In another related research effort, Petre observed that models are useful for requirements elicitation where "*stakeholders tend to be highly technical*" and "*then discard the UML diagrams*" [181]. These circumstances reflect the situation examined in this thesis well, as most stakeholders undertake development tasks ("*highly technical*") and the timestamps of the models indicate a comparable procedure ("*then discard the UML diagrams*", see also Chapter 4). However, using models in this manner has been considered to provide only "*limited value*" by Selic [211]. For this reason, it seems advantageous to move towards "*fully formal models*" to allow for simulation or generation of source code and test cases, although the latter is possible to a certain degree even now (discussed in Chapter 6). However, this may reach far beyond what is originally intended by the examined approach and it must be carefully considered if this is worth the effort.

Such formal models might be within reach considering the pre-existing information in the RS. One of the insights in Chapter 3 suggests that the textual representation is, among other things, used to accommodate details or further information. While this data might be better presented in a textual manner, it remains open to discussion why it is not also integrated into graphical models (i.d., the activity diagram). Some of these details might be suitable for enriching the activities in a way that enables simulation or the generation of source code and test cases.

Unfortunately, such enrichment of activity diagrams cannot be fully automated unless the textual information exists in a well-defined form (as such the text would constitute a model of its own). On the contrary, it should be expected that extensive manual work will be required to ensure a proper outcome [212]. This is due to the fact that in the case examined in this thesis, the additional text also contains elements in NL text. Thus, a proper integration of content in this form requires manual work which might be facilitated by sophisticated approaches (e.g., model-to-requirements approaches, see Subsection 2.4.2). This propagation of information back to the model can be regarded as round trip engineering which represents another area of interest in MDD research [213]. Nevertheless, doubt remains as to whether practitioners see this as desirable and whether they can make this work in a way that justifies the additional costs associated with such a procedure. According to the findings of Petre, this seems unrealizable in an industrial setting [182]. Also, further challenges

lie in the identification of "suitable" textual information itself. Signal names might be appropriate candidates for integration into the model while descriptive information might not be applicable.

At the same time, in what way this additional information should be incorporated into the activity diagram remains an open issue. More formal models require notations that are more formal than NL text (e.g., for the examined case Action Language for Foundational UML (ALF) might be appropriate [174]). However, these more formal notations are not even remotely as accessible to different audiences as NL text. An additional benefit of the integration of unique textual information into the activity diagram is the prevention of fragmentation (scattering of information across multiple sources). Fragmentation itself may complicate usability and lead to inconsistencies.

### 7.2.3.2 A Case Study on Quality Issues and Inconsistencies

Petre identified consistency issues as one of the most frequently mentioned reasons why UML is not adopted in practice [182]. While this thesis finds that inconsistencies have a significant impact on the use of models for specification purposes, the assumption that consistency issues all together prevent the adoption of MDD practices or UML is not confirmed in the context of this thesis. Moreover, the defined categories of inconsistencies and other quality issues indicate that the different characteristics of the categories have a varying impact on the quality of the specification artifacts. This is of importance, as the use of multiple representations is inherently prone to inconsistencies – especially if maintained independently by different stakeholders [191]. In cases where the models are only used at one stage and never updated, inconsistencies are inevitable. Hence, it comes as no surprise that comparable specification approaches can be expected to suffer from inconsistencies as well. Although different implementations may use other kinds of diagram types, similar categories of inconsistencies are likely to appear.

For example, in their research, van der Aa et al. examine an instance of such an approach, which uses BPMN as the graphical representation with a textual description to model processes [2]. In their work, the authors attempt to detect inconsistencies between these representations. They identify two kinds of inconsistencies: missing activities in either of the representations and conflicting order of process steps. These inconsistencies describe the same properties as the categories **Missing Element/Object** and **Wrong Placement** in this thesis. Still, the inconsistencies **Incorrect Logic** and **Textual Differences** could also be applicable to van der Aa et al.'s examined approach. Only the category **Wrong Type** is not applicable, since their textual description does not include attributes, meaning there is no explicit type existent in the textual descriptions.



The applicability of many of the inconsistency categories stems from the fact that activity diagrams and [BPMN](#) both describe behavior using some kind of object or information flow. Nevertheless, the categories can also be mapped to other behavioral notations such as state charts. In that case **Missing Element/Object** and **Textual Differences** are applicable. Whether **Incorrect Logic** and **Wrong Placement** are applicable is open to interpretation to a certain degree. For state charts, **Incorrect Logic** could represent differences in the propositional logic expressions of the transitions between the states. Also, assuming a different notion of logic, the category **Incorrect Logic** could imply differences concerning which states are connected to one another, although this could also be interpreted as a kind of **Wrong Placement** in a broader sense (i.d., conflicting order of predecessors in a graph like interpretation). For the category **Wrong Type**, its applicability for state charts again depends on whether the textual representation features attributes and includes a concept of types.

Aside from behavioral notations, similar categories may appear in other model types as well. For instance, block definition diagrams of [SysML](#) and a textual representation may exhibit the categories **Missing Element/Object** and **Textual Differences**. For **Incorrect Logic** and **Wrong Placement**, the situation is similar to what is described for state charts. Differing connections can be seen as a difference in the underlying logic of the diagram or as a type of conflicting order. Again, **Wrong Type** is at the very least connected to the kind of textual representation used and hence may or may not be applicable.

#### 7.2.3.3 *Information Extraction from High-Level Activity Diagrams*

In his paper "Modern Requirements Specification" from 2003, Fire-smith notes the importance of automatic specification generation for improving requirements specification tasks [82]. Because of the wide use of models in [RE](#), this has lead to a number of works employing models for the generation of requirements artifacts (see also: Subsection 2.4.1). Nicolás and Toval review the works on the generation of requirements artifacts from models [165]. They conclude by recommending five properties that such generative techniques must exhibit:

1. Automatable technique
2. Resembling structure between representations
3. Enable multiple textual views
4. Modifiable output
5. Synchronization



First, Nicolás and Toval state that the technique should “*enable the automatic or closely monitored generation*”. This need is met in this thesis by the technique presented in Chapter 6, as it is implementable in a fully automatic manner (see Appendix C).

Second, they advise that the model and the aligned textual representation should follow the same structure. This property primarily aims at the alignment of models with the entire document structure. The presented technique in this thesis, however, produces a textual representation for functions and not entire RS documents and hence this recommendation is only partially applicable. But even if one demands that the property should also hold for distinct functions, it contradicts the next (third) property to a certain degree.

The third property relates to the ability of tailoring the textual representation to different target readerships. This is logical, as different stakeholders have different preferences. This may mean that some stakeholders only need certain information and may not be interested in all the aspects of the model. For instance, stakeholders interested in signal names do not necessarily need information about execution sequences. This is in violation of the second property, since a simple list does no longer resemble the structure of a behavioral model. Therefore, the second property should be directed at the form of generated text rather than the generative technique itself so long as it is possible to generate multiple forms of text. The presented technique in this thesis extracts information from activity diagrams. As a result, it is not restricted to a pre-defined form and can be used flexibly to generate any form of text required – some of which may resemble the structure of the original model. Hence, both the second and third properties can be considered fulfilled by the technique presented in this thesis.

Fourth, Nicolás and Toval emphasize that the generated format should be modifiable. The technique presented in this thesis merely reflects a possibility that can ultimately be used to produce any form of output. Its prototypical implementation (presented in Appendix C) generates the textual representation in IBM DOORS. This format is modifiable and thus fulfills this requirement.

Finally, they argue that once the documentation is generated it should be possible to enable synchronization. At first glance, this property appears to enable the modification of requirements artifacts by multiple stakeholders while still maintaining consistency across the artifacts. However, our findings from Chapter 3 show that stakeholders consider this functionality harmful and prefer that changes be incorporated at a single point. Moreover, synchronization between multiple views might not be possible in any direction. Due to the fact that not all forms of text necessarily contain all information (see also Nicolás and Toval’s third recommendation). Hence, it is not always possible to correctly propagate changes beyond textual modifications

back to the activity diagram correctly (e.g., changes in propositional logic) – a capability that is included by the prototypical implementation.

Thus, the presented technique in this thesis meets the five recommendations made by Nicolás and Toval for generative techniques.

## OUTLOOK

---

There are several issues that have not been examined in this thesis. The main goal of this thesis was to investigate an implementation of a specification approach in practice and address its issues and challenges. However, the investigation and the solutions proposed are limited in scope. The limitations of each contribution have already been discussed as part of the publications in the respective chapters and already highlight a number of research topics whose further examination may yield an improvement in understanding or efficiency.

### ANALYSIS OF OTHER IMPLEMENTATIONS

As a whole, this thesis is focused on the analysis and the problems of a single case. To improve validity as well as the generalizability of the findings, future work should engage in the analysis of different implementations of such specification approaches in industry. As every development project is different and organizational conditions have a significant impact, it is to be expected that these implementations will differ from the specific case in this thesis.

Technical considerations may lead to the application of other types of models or even modeling languages and a different form of textual representation. This may affect the overall perception of such an approach, but could also lead to new types of inconsistencies and other quality issues, since these are bound more to the types of representations used than the underlying idea of coexisting development artifacts. At the very least, other types of models require new techniques to enable automatic generation of textual representation. However, unless these combinations are in fact required in practice, it seems unnecessary to add to the already existing assortment of techniques (see Subsection 2.4.1). Nevertheless, such work may reveal new challenges specific to certain model types that must be addressed first (analogous to the redundancy removal in Chapter 5). This could provide insights and applications for these model types beyond simply enabling the generation of textual representations.

Aside from the technical considerations, different social circumstances also impact the adoption and use of comparable specification approaches. Gaining further insights into the surrounding conditions may allow for the identification of general success factors that influence possible benefits and drawbacks.

## ENTITY MATCHING

For the case analyzed in this thesis, it is recommended to begin the requirements specification with the activity diagrams and generate the textual representations from these diagrams (i.a., see Chapter 3). While this addresses a number of problems of the approach examined (e.g., inconsistencies), it does not guarantee that the representations will remain consistent. In the course of day-to-day business, it is possible that the representations will become inconsistent and traceability maintenance will be neglected. Also, the informal adoption of MDD practices may lead to a situation in which graphical and textual representations coexist, but are not properly integrated. In this situation, both representations may still be consistent but no traceability is established. To support continued consistency and improve traceability, matching approaches could be applied. Matching has received attention for a number of applications (i.a. to relate elements of different models to one another [66]). For process models and their textual descriptions this has already been demonstrated by van der Aa et al. [2], who aim to detect inconsistencies in this manner. While they mainly rely on linguistic approaches, structural peculiarities of representations could also be utilized. For the case examined in this thesis, structural properties that express the execution order or propositional logic relations appear to be a promising point for improving the results of a matching. This way, the additional inconsistencies defined in this thesis and those not considered by van der Aa et al. could also be automatically detected.

## FURTHER REPRESENTATIONS

Gross and Doerr have already argued for multiple views of requirements [100]. On the textual level this thesis addresses this need for tailored views by offering multiple possible textual representations for the activity diagrams (see Appendix B). However, there might be alternatives to textual descriptions. For instance, Petre reports that in practice, it is common to integrate UML diagrams with other graphical notations such as entity-relationship diagrams or BPMN [181]. Such coexistence between multiple graphical notations faces similar challenges (e.g., inconsistencies), and might also need to contend with yet undiscovered problems. As such, it is an area of interest for further investigation.

Aside from the coexistence aspect, it may prove fruitful to experiment with existing models in a more sophisticated way. For instance, Gemino compared graphical models and animated models [91]. While he found no measurable benefit for animation, he admits that a very crude form of animation was used and hence more work is required to further investigate this type of model use.

## Part IV

### APPENDIX

The following three appendices contain further information on the topics within this thesis.

- Appendix [A](#) provides formal definitions for the activity diagrams and textual representations of this thesis and uses these to formally define the categories of inconsistencies and other quality issues in Chapter [4](#).
- Appendix [B](#) presents alternative textual representations of the activity diagrams of the examined use case and an evaluation thereof.
- Appendix [C](#) presents a tool prototype that implements the concepts presented in Chapter [5](#) and Chapter [6](#) in order to automatically generate textual representations of activity diagrams.



## FORMALIZATION OF INCONSISTENCIES AND QUALITY ISSUES

---

This appendix presents the formalized criteria of the categories presented and assessed in Chapter 4. For these definitions, the main challenge lies in the mismatch between the different underlying structures of models and RS documents [28]. At first, a construct for UML2 activities (directed graph) is presented that is used for the definition of the categories. Then follows a formal definition for the textual representations (tree structure) used by the industry partner (for a sample see Chapter 3 and Chapter 4). These definitions are employed to formally define the categories. The following terminology is used in this appendix.

### TERMINOLOGY

**Activity:** The model as defined by the OMG UML2 Specification Version 2.5 [172, p. 371].

**Element:** An *ExecutableNode* or a *ControlNode* in the *Activity*.

**Activity Graph:** The construct that is used for *Activities* in the following to define the categories of inconsistencies and other quality issues.

**Element Vertex:** Counterpart of an *ActivityNode* in an *Activity* in the *Activity Graph*.

**Element Constituent:** Individual propositional logic statement within an Element Vertex.

**Textual Representation:** The corresponding text of an *Activity* in an RS.

**Object:** A row in a Textual Representation.

**Textual Structure:** The construct used for a Textual Representation in the following to define the categories of inconsistencies and other quality issues.

**Object Vertex:** Counterpart of a row in a Textual Representation in the *Textual Structure*.

**Object Constituent:** Individual propositional logic statement within an Object Vertex.

#### A.1 FORMAL SEMANTICS OF THE EXAMINED UML2 ACTIVITIES

The assumed semantics of **UML2 activities** in this section deviate from those presented in Chapter 6. The reason for this is, that in Chapter 6, it is assumed that the inputs are *Activities* with unique atomic elements. This assumption is guaranteed by the method presented in Chapter 5. However, this assumption does not hold for the *Activities* encountered at the industry partner. As a result, it is necessary to extend the definition of *Activity Graphs* in Chapter 6 in order to include all aspects of the inconsistencies and other quality issues. For this purpose, an **UML2 activity** is seen as a construct named *Activity Graph*. The *ActivityNodes* of an *Activity* are vertices in the *Activity Graph*.

##### **Definition 1 Activity Graph**

An Activity Graph  $A$  is defined as

$$A \stackrel{\text{def}}{=} (V_A, V_{\text{Intra}_A}, \text{succ}_A, \text{type}_A, \text{label}_A, \text{content}_A, \text{logic}_A) \quad (\text{A.1})$$

where

1.  $V_A$  is a non-empty set of element vertices that represents the *ActivityNodes* of the Activity,
2.  $V_{\text{Intra}_A}$  is a set of element constituents that represents propositional logic statements of an *ActivityNode*,
3.  $\text{succ}_A : V_A \rightarrow \mathcal{P}(V_A)$  is the successor function that assigns  $v_a \in V_A$  its set of successors,
4.  $\text{type}_A : V_A \rightarrow \{\text{Trigger}, \text{Check}, \text{Function}, \text{Merge}, \text{Decision}, \text{Join}, \text{Fork}, \text{End}\}$  is the function that assigns each element vertex a type,
5.  $\text{label}_A : V_A \rightarrow \Sigma^*$  is a function that assigns each element vertex the label of its *ActivityNode* ( $\Sigma$  is an arbitrary labeling alphabet),
6.  $\text{content}_A : V_A \rightarrow \mathcal{P}(V_{\text{Intra}_A})$  is the function that assigns each element vertex the set of its element constituents, and
7.  $\text{logic}_A : V_A \rightarrow \{\text{AND}, \text{OR}\}$  is the function that assigns each element vertex the propositional logic operator of its element constituents.

The *Activity Graph* is a graph that corresponds to the *Activity*. The elements of the *Activity* match the element vertices of the graph. These element vertices in turn consist of element constituents. Element constituents are individual propositional logic statements within an element vertex. The element constituents are assigned to each element



vertex by the function  $content_A$ . *ControlNodes* (e.g., *MergeNodes*, *JoinNodes*) do not have element constituents ( $content_A$  assigns the empty set). In case an element vertex consists of more than one element constituent, they are connected by an AND or OR-operator. Which operator it is, is determined by the function  $logic_A$ . Each element vertex may have a set of successors. In the *Activity Graph* the element vertices are successors that correspond to the *Target* of outgoing edges in the *Activity*. Every element vertex has one of the following types: *Trigger*, *Check*, *Function*, *Merge*, *Decision*, *Join*, *Fork*, *End*. For the element vertices that correspond to *ControlNodes*, the type equals the kind of *ControlNode*. The *ExecutableNodes* may either be *Trigger*, *Check*, or *Function*. Finally, each element vertex has a label which corresponds to the text of the element in the activity. These labels consist of propositional logic statements as well as their operators. *ControlNodes* do not have labels (i.d.,  $label_A$  assigns the empty word  $\epsilon$ ).

## A.2 FORMAL DEFINITION OF THE TEXTUAL REPRESENTATION

RS documents are hierarchically structured documents [119]. This structure can be represented by an ordered tree [28, 229]. The document tree of the excerpt shown in Figure 2.1 is depicted in Figure A.1.

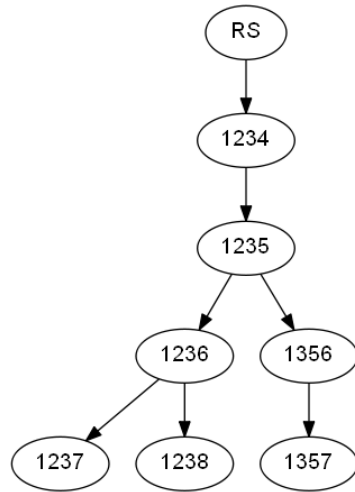


Figure A.1: Document tree of RS in Figure 2.1

The vertex on the top of the tree (root vertex) represents the document. The other vertices are referenced by their *ID* in the excerpt. The *Level* in the document represents the number of edges of a vertex from its position to the vertex that embodies the document (root vertex). Since the RS document comprises a number of other characteristics, a pure ordered tree does not cover all the properties. Therefore a construct for the textual representation is used that looks as follows:

**Definition 2 Textual Structure**

A Textual Structure  $T$  is defined as an ordered tree

$$T \stackrel{\text{def}}{=} (V_T, V_{\text{Intra}_T}, O_{\text{Out}}, O_{\text{In}}, \text{parent}_T, \leq, \text{type}_T, \text{label}_T, \text{content}_T, \text{logic}_{\text{Out}}, \text{logic}_{\text{In}}, \llbracket \rrbracket_{\text{Out}}, \llbracket \rrbracket_{\text{In}}) \quad (\text{A.2})$$

where

1.  $V_T$  is a non-empty set that represents the rows of a textual representation,
2.  $V_{\text{Intra}_T}$  is a set of object constituents that represents the individual propositional logic statements of an object in a textual representation,
3.  $O_{\text{Out}}$  is a set of propositional logic operators that represent the connections between object vertices,
4.  $O_{\text{In}}$  is a set of propositional logic operators that represent the connections within an object vertex between object constituents,
5.  $\leq: V_T \times V_T$  is a partial order relation,
6.  $\text{parent}_T: V_T \rightarrow V$  is a function that assigns each object vertex its parent vertex,
7.  $\text{type}_T: V_T \rightarrow \{\text{Trigger}, \text{Check}, \text{Function}\}$  is a function that assigns each object vertex a type,
8.  $\text{label}_T: V_T \rightarrow \Sigma^*$  is a function that assigns each object vertex the label of its object ( $\Sigma$  is an arbitrary labeling alphabet),
9.  $\text{content}_T: V_T \rightarrow \mathcal{P}(V_{\text{Intra}_T})$  is a function that assigns each object vertex the set of its object constituents,
10.  $\text{logic}_{\text{Out}}: V_T \rightarrow O_{\text{Out}}$  is a function that assigns each object vertex its outer propositional logic operator  $o_{\text{Out}} \in O_{\text{Out}}$ ,
11.  $\text{logic}_{\text{In}}: V_T \rightarrow O_{\text{In}}$  is a function that assigns each object vertex its inner propositional logic operator  $o_{\text{In}} \in O_{\text{In}}$ ,
12.  $\llbracket \rrbracket_{\text{Out}}: O_{\text{Out}} \rightarrow \{\text{AND}, \text{OR}, \_ \}$  is a function that assigns an outer propositional logic operator its semantics, and
13.  $\llbracket \rrbracket_{\text{In}}: O_{\text{In}} \rightarrow \{\text{AND}, \text{OR}, \_ \}$  is a function that assigns an inner propositional logic operator its semantics.

The *Textual Structure* consists of object vertices  $V_T$ . These in turn may consist of multiple object constituents. Same as in the *Activity Graph* the object constituents represent individual propositional logic statements. The structure of the tree is realized by the function  $\text{parent}_T$  and its order by the partial order relation  $\leq$  (complete order relation for sibling vertices). The object constituents are connected via

the inner operators  $O_{In}$ . The object vertices may be connected to following object vertices on the same level by the outer operators  $O_{Out}$ . The function  $logic_{Out}$  assigns the outer operator to an object vertex. The function  $logic_{In}$  does the same for the inner operators. In both cases these operators can be AND or OR-operators. In case there is no following object vertex on the same level or there is only one object constituent, the respective operators are denoted with  $_$ . The meaning of the operators is determined by the function  $[[[Out]$  or  $[[[In]$ , respectively. Finally, each object vertex has a type which equals the value in the *type* attribute of the *RS* and is derived from the *Activity*.

### A.3 CONNECTIONS BETWEEN THE ACTIVITY DIAGRAM AND THE TEXTUAL REPRESENTATION

A number of relations can be found between the activity diagrams and their corresponding textual representations. As can be seen in the sample in Chapter 3 and Chapter 4, elements in the activity diagram and objects in the textual representation relate to one another. These relations also apply to the *ControlNodes* in the activity diagram which can be followed to the operators in the textual representation. To express these relations the construct *correspondent* is used which connects the entities of the *Activity Graph* with those of the *Textual Structure*.

#### Definition 3 *correspondent*

$$correspondent \subseteq (V_{Intra_A} \times V_{Intra_T}) \cup (V_A \times (O_{out} \cup O_{in})) \quad (A.3)$$

The relation *correspondent* contains pairs of:

1. element constituents and object constituents, and
2. element vertices and outer or inner operators.

Item 1. represents the relation of a propositional logic statement within an *ExecutableNode* to a propositional logic statement within an object. Item 2. represents the relation of a *ControlNode* to an AND or OR-Operator in the text attribute of an object.

In case there are correspondences between the specification artifacts, it is desirable to link them to one another (see Section 2.1). The existence of such a link is expressed by the relation *trace*.

#### Definition 4 *trace*

$$trace \subseteq V_A \times V_T \quad (A.4)$$

The relation *trace* contains pairs of element vertices and object vertices for which an information exists to assign them to one another. The trace links are realized for single elements and objects, as they are the smallest entities within the respective tools that are uniquely identifiable.

In summary, *trace* represents the connections between uniquely identifiable entities, while *correspondent* represents connections on a even more fine-grained level (e.g., between corresponding information within the entities).

#### A.4 CATEGORIES OF INCONSISTENCIES AND OTHER QUALITY ISSUES

This section contains the definitions of the identified inconsistencies and other quality issues of Chapter 4. For each, an informal definition is provided as well as a formal definition that is also expressed as a formula. Each formula describes the set of all instances of the category.

##### *Missing Tracing*

##### *Informal Definition*

In an activity diagram with corresponding textual representation exist elements which possess corresponding objects in the textual representation. In case there is no information or possibility available to relate them to one another correctly, the category *Missing Tracing* is existent.

##### *Formal Definition*

For an *Activity Graph A* and a *Textual Structure T*, whose constituents are connected by the relation *correspondent*, there is no information available on this connection in the relation *trace*. Expressed as a formula:

##### **Definition 5** *Missing Tracing*

$$\begin{aligned} \text{Missing Tracing} := \{ & (v_A, v_T) \in V_A \times V_T : \\ & (v_A, v_T) \notin \text{trace} \wedge \exists (v_{\text{Intra}_A}, v_{\text{Intra}_T}) \in \text{correspondent} : \\ & v_{\text{Intra}_A} \in \text{content}_A(V_A) \wedge v_{\text{Intra}_T} \in \text{content}_T(V_T) \} \end{aligned} \quad (\text{A.5})$$

*Missing Element/Object**Informal Definition*

In case a textual representation contains an object for which there is no corresponding element in the corresponding activity diagram, the category *Missing Element* is existent. In case an activity diagram contains an element for which there is no corresponding object in the corresponding textual representation, the category *Missing Object* is existent. Both categories together result in the category *Missing Element/Object*.

*Formal Definition*

There is an element constituent within an element in *Activity Graph A* that does not have a counterpart object constituent in the *Textual Structure T*. Expressed as a formula:

**Definition 6 *Missing Object***

$$\begin{aligned} \text{Missing Object} &:= \{v_A \in V_A : \\ &\nexists (v_{\text{Intra}_A}, v_{\text{Intra}_T}) \in \text{correspondent} : \\ &v_{\text{Intra}_A} \in \text{content}_A(v_A)\} \end{aligned} \quad (\text{A.6})$$

There is an object constituent within an object in *Textual Structure T* that does not have a counterpart element constituent in the *Activity Graph A*. Expressed as a formula:

**Definition 7 *Missing Element***

$$\begin{aligned} \text{Missing Element} &:= \{v_T \in V_T : \\ &\nexists (v_{\text{Intra}_A}, v_{\text{Intra}_T}) \in \text{correspondent} : \\ &v_{\text{Intra}_T} \in \text{content}_T(v_T)\} \end{aligned} \quad (\text{A.7})$$

The category *Missing Element/Object* results from *Missing Element* and *Missing Object* combined.

**Definition 8 *Missing Element/Object***

$$\begin{aligned} \text{Missing Element/Object} &:= \text{Missing Element} \cup \\ &\text{Missing Object} \end{aligned} \quad (\text{A.8})$$

*Wrong Placement**Informal Definition*

In case an element in the activity diagram has a corresponding object in the textual representation, but their placement in the respective

representations deviate, then the category *Wrong Placement* is existent. A deviation in placement is existent if the successors of the element and the object differ.

#### Formal Definition

The category *Wrong Placement* may appear for an element vertex in an *Activity Graph A* and its corresponding object vertex in the *Textual Structure T* and encompasses three different cases. Those three cases result from the different ways successors in the *Activity Graph* can be represented in the *Textual Structure*.

First case: the successor of an element vertex in *ActivityGraph A* is of type *Join* or *Merge* and is not represented by the correct inner operator. Expressed as a formula:

#### Definition 9 Case 1

$$\begin{aligned}
 \text{Case 1} &:= \{(v_A, v_T, o_{in}) \in V_A \times V_T \times O_{in} : \\
 &\exists v_{A_{suc}} \in \text{suc}_A(v_A) : \text{type}_A(v_{A_{suc}}) \in \{\text{Merge}, \text{Join}\} \wedge \\
 &\exists v_{Intra_A} \in \text{content}_A(v_A), v_{Intra_T} \in \text{content}_T(v_T) : \\
 &(v_{Intra_A}, v_{Intra_T}) \in \text{correspondent} \wedge \\
 &(v_{A_{suc}}, o_{in}) \in \text{correspondent} \wedge o_{in} \neq \text{logic}_{in}(v_T)\}
 \end{aligned} \tag{A.9}$$

Second case: the successor of an element vertex in *Activity Graph A* is of type *Join* or *Merge* and is not represented by the correct outer operator. The outer operator may belong to the object vertex of the corresponding element or to the predecessor of the object vertex in the *Textual Structure T*. Expressed as a formula:

#### Definition 10 Case 2

$$\begin{aligned}
 \text{Case 2} &:= \{(v_A, v_T, o_{out}) \in V_A \times V_T \times O_{out} : \\
 &\exists v_{A_{suc}} \in \text{suc}_A(v_A) : \text{type}_A(v_{A_{suc}}) \in \{\text{Merge}, \text{Join}\} \wedge \\
 &\exists v_{Intra_A} \in \text{content}_A(v_A), v_{Intra_T} \in \text{content}_T(v_T) : \\
 &(v_{Intra_A}, v_{Intra_T}) \in \text{correspondent} \wedge \\
 &(v_{A_{suc}}, o_{out}) \in \text{correspondent} \wedge \\
 &\neg(o_{out} = \text{logic}_{out}(v_T) \vee \\
 &(\exists v_{T_{pred}} \in V_T : v_{T_{pred}} \leq v_T \wedge o_{out} = \text{logic}_{out}(v_{T_{pred}})))\}
 \end{aligned} \tag{A.10}$$

Third case: the successor of an element vertex in *Activity Graph A* is of type *Check*. In this case the successor in the *Textual Structure T* must be placed one level below. Otherwise it is a wrong placement of the element/object. Expressed as a formula:

**Definition 11 Case 3**

$$\begin{aligned}
\text{Case 3} := & \{(v_A, v_T) \in V_A \times V_T : \\
& \exists v_{A_{suc}} \in \text{suc}_A(v_A) : v_{\text{Intra1}_A} \in \text{content}_A(v_{A_{suc}}) \wedge \\
& \exists v_{T_{parent}} \in \text{parent}(v_T) : v_{\text{Intra1}_T} \in \text{content}_T(v_{T_{parent}}) \wedge \\
& \exists v_{\text{Intra2}_A} \in \text{content}_A(v_A), v_{\text{Intra2}_T} \in \text{content}_T(v_T) : \\
& (v_{\text{Intra2}_A}, v_{\text{Intra1}_T}) \in \text{correspondent} \wedge \\
& (v_{\text{Intra1}_A}, v_{\text{Intra2}_T}) \notin \text{correspondent}\}
\end{aligned} \tag{A.11}$$

The combination of these cases constitutes the category *Wrong Placement*.

**Definition 12 Wrong Placement**

$$\text{Wrong Placement} := \text{Case 1} \cup \text{Case 2} \cup \text{Case 3} \tag{A.12}$$

The three cases do not cover all possibilities of successor relations in *Activities*. The presented cases are merely those that in fact appeared in the [RS](#) documents of the industry partner. Other cases would be more fictional in nature. These other cases include situations that are syntactically correct in the sense of the [UML2](#) specification, but do not adhere to the pattern of the activity diagrams of the industry partner (see Chapter [3](#) and Chapter [4](#)). As there are various possibilities to construct the activities in a way that do not fit the pattern, not all possibilities are predictable. At the same time, these definitions would provide no benefit as the representations are inherently inconsistent if the pattern is not used. Thus a formal definition would be futile.

*Incorrect Logic**Informal Definition*

Basically the category *Incorrect Logic* refers to situations in which the underlying propositional logic expression of the activity diagram and the textual representation differ. As this might be the case if the categories *Missing Element/Object* and *Wrong Placement* appear, *Incorrect Logic* does not include their criteria. This ensures a unique definition of the category *Incorrect Logic*. Hence, the category *Incorrect Logic* is existent in these cases:

1. Ambiguous Syntax: it is not possible to unambiguously derive a propositional logic expression because of syntax errors in the textual representation (i.e. the textual representation does not adhere to the pattern presented in Chapter [3](#)).
2. Wrong Operator: the *ControlNodes* in the activity diagram do not match the operator in the textual representation.

There may be further constellations that lead to different underlying propositional logic statements between the representations. Nevertheless, the completeness of the category is not demonstrated, as the listed properties can be considered the important ones, since they encompass all situations that occurred in the examined system.

#### *Formal Definition*

Ambiguous syntax exists if there are object vertices that have a following object vertex on the same level, but do not possess an outer operator. This criterion also includes object vertices that do not have a following object on the same level and whose outer operator is not  $\_$ . Multiple object vertices all without child vertices also belong to this criterion. As their contents should be grouped into a single object vertex – following the pattern presented in Chapter 3. Also, it would be possible to connect multiple vertices of the same level by different operators. Without knowledge which operators binds stronger, it is not possible to derive a propositional logic expression unambiguously.

#### **Definition 13** *Ambiguous Syntax*

$$\begin{aligned}
 \text{Ambiguous Syntax} := & \{(v_T, o_{Out}) \in V_T \times O_{Out} : \\
 & o_{Out} = \text{logic}_{Out}(v_T) \wedge \\
 & ((\exists v_{T_{suc}} \in V_T : v_T < v_{T_{suc}} \wedge \llbracket o_{Out} \rrbracket_{Out} = \_) \vee \\
 & (\nexists v_{T_{suc}} \in V_T : v_T < v_{T_{suc}} \wedge \llbracket o_{Out} \rrbracket_{Out} \in \{AND, OR\})) \vee \\
 & (\exists v_{T_2} \in V_T : v_T < v_{T_2} \wedge \\
 & \text{parent}_T^{-1}(v_{T_1}) = \emptyset \wedge \text{parent}_T^{-1}(v_{T_2}) = \emptyset))\}
 \end{aligned} \tag{A.13}$$

$\text{parent}_T^{-1}$  is the inverse function of  $\text{parent}_T$  and defined as:

$$\text{parent}_T^{-1}(v_p) := \{v_c : \text{parent}(v_c) = v_p\} \text{ with } v_c, v_p \in V_T$$

A *Wrong Operator* exists, in case an element vertex in *Activity Graph A* of type *Join* corresponds to an outer or inner operator in the *Textual Structure T* that possesses the meaning *OR*. The criterion also includes the opposite case: an element vertex in *Activity Graph A* of type *Merge* corresponds to an outer or inner operator in the *Textual Structure T* that possesses the meaning *AND*.



**Definition 14** *Wrong Operator*

$$\begin{aligned}
\text{Wrong Operator} := \{ & v_A \in V_A : \left( \text{type}_A(v_A) = \text{Merge} \wedge \right. \\
& ((\exists o_{Out} \in O_{Out} : (v_A, o_{Out}) \in \text{correspondent} \wedge \\
& \llbracket o_{Out} \rrbracket_{Out} = \text{AND}) \vee \\
& (\exists o_{In} \in O_{In} : (v_A, o_{In}) \in \text{correspondent} \wedge \\
& \llbracket o_{In} \rrbracket_{In} = \text{AND})) \vee \\
& \left. \left( \text{type}_A(v_A) = \text{Join} \wedge \right. \right. \\
& ((\exists o_{Out} \in O_{Out} : (v_A, o_{Out}) \in \text{correspondent} \wedge \\
& \llbracket o_{Out} \rrbracket_{Out} = \text{OR}) \vee \\
& (\exists o_{In} \in O_{In} : (v_A, o_{In}) \in \text{correspondent} \wedge \\
& \left. \left. \llbracket o_{In} \rrbracket_{In} = \text{OR}) \right) \right) \}
\end{aligned} \tag{A.14}$$

The two criteria *Ambiguous Syntax* and *Wrong Operator* combined represent the category *Incorrect Logic*.

**Definition 15** *Incorrect Logic*

$$\text{Incorrect Logic} := \text{Ambiguous Syntax} \cup \text{Wrong Operator} \tag{A.15}$$

*Textual Differences**Informal Definition*

The label of an element in the activity diagram differs from the label of the corresponding object in the corresponding textual representation.

*Formal Definition*

An element vertex in the *Activity Graph*  $A$  and its corresponding object vertex in the *Textual Representation*  $T$  do not have the same label. Expressed as a formula:

**Definition 16** *Textual Differences*

$$\begin{aligned}
\text{Textual Differences} := \{ & (v_A, v_T) \in V_A \times V_T : \\
& \exists (v_{Intra_A}, v_{Intra_T}) \in \text{correspondent} : \\
& v_{Intra_A} \in \text{content}_A(v_A) \wedge v_{Intra_T} \in \text{content}_T(v_T) \wedge \\
& \text{label}_A(v_A) \neq \text{label}_T(v_T) \}
\end{aligned} \tag{A.16}$$

### *Wrong Type*

#### *Informal Definition*

In case the type of an element in the activity diagram does not match the type of the object in the textual representation, the category *Wrong Type* is existent. In a broader sense this category resembles the non-canonicity pattern *Implicit Action* as defined by Leopold et al. [134]. In contrast to their understanding, it is not an inconsistency between a model element and a describing text (in their case its label), but an inconsistency between a model element and an explicitly modeled property of the textual description.

#### *Formal Definition*

There is an element vertex in the *Activity Graph A* whose type is *Trigger*, *Check*, or *Function* and whose corresponding object in the *Textual Representation T* does not have the same type. Expressed as a formula:

#### **Definition 17** *Wrong Type*

$$\begin{aligned}
 \text{Wrong Type} &:= \{(v_A, v_T) \in V_A \times V_T : \\
 &\text{type}_A(v_A) \in \{\text{Trigger}, \text{Check}, \text{Function}\} \wedge \\
 &\exists (v_{\text{Intra}_A}, v_{\text{Intra}_T}) \in \text{correspondent} : \\
 &v_{\text{Intra}_A} \in \text{content}_A(v_A) \wedge v_{\text{Intra}_T} \in \text{content}_T(v_T) \wedge \\
 &\text{type}_A(v_A) \neq \text{type}_T(v_T)\}
 \end{aligned} \tag{A.17}$$

### *Unnecessary Repetition*

#### *Informal Definition*

In case there are two strings in the textual representation that are derived from one element in the activity diagram, the category *Unnecessary Repetition* is existent.

#### *Formal Definition*

In an *Activity Graph A* there is an element constituent that is part of two pairs in the relation *correspondent*. Expressed as a formula:

#### **Definition 18** *Unnecessary Repetition*

$$\begin{aligned}
 \text{Unnecessary Repetition} &:= \{v_a \in V_A : \\
 &\exists v_{\text{Intra}_A} \in V_{\text{Intra}_A}, v_{\text{Intra}_T} \in V_{\text{Intra}_T} : \\
 &v_{\text{Intra}_A} \in \text{content}_A(v_a) \wedge \\
 &|(v_{\text{Intra}_A}, v_{\text{Intra}_T}) \in \text{correspondent}| > 1\}
 \end{aligned} \tag{A.18}$$

*Redundant Elements**Informal Definition*

In the activity diagram exists more than one element with the same label.

*Formal Definition*

In an *Activity Graph*  $A$  exists more than one element vertex with the same label. Expressed as a formula:

**Definition 19** *Redundant Elements*

$$\begin{aligned} \text{Redundant Elements} := \{v_{A1}, v_{A2} \in V_A : \\ v_{A1} \neq v_{A2} \wedge \text{label}_A(v_{A1}) = \text{label}_A(v_{A2})\} \end{aligned} \quad (\text{A.19})$$

*Non Atomic Element/Object**Informal Definition*

The category *Non Atomic Element/Object* is existent, in case there is an element in the activity diagram (*Non Atomic Element*) or an object in the textual representation (*Non Atomic Object*) that contains multiple propositional logic statements. *Non Atomic Element* follows the definition of atomicity in models by Pittke which states that the modeling language is used for separation instead of natural language [184, p. 90]. Hence, *Non Atomic Object* addresses the characteristic of a requirement being *singular* [226].

*Formal Definition*

In an *Activity Graph*  $A$  exists an element that consists of more than one element constituent.

**Definition 20** *Non Atomic Element*

$$\text{Non Atomic Element} := \{v_A \in V_A : |\text{content}_A(v_A)| > 1\} \quad (\text{A.20})$$

In a *Textual Structure*  $T$  exists an object vertex that consists of more than one object constituent.

**Definition 21** *Non Atomic Object*

$$\text{Non Atomic Object} := \{v_T \in V_T : |\text{content}_T(v_T)| > 1\} \quad (\text{A.21})$$

Both criteria combined represent the category *Non Atomic Element/Object*.

**Definition 22** *Non Atomic Element/Object*

$$\text{Non Atomic Element/Object} := \text{Non Atomic Element} \cup \text{Non Atomic Object} \quad (\text{A.22})$$

## ALTERNATIVE TEXTUAL REPRESENTATIONS OF ACTIVITY DIAGRAMS

---

This appendix presents alternatives to the textual representation used by the industry partner (see Chapter 3 and Chapter 4) as well as an assessment of users. The textual representation in Chapter 3 and Chapter 4 is referred to as the *Original Representation* and is part of the user assessment. The contents of this appendix are partly based on a previous publication [24].

### B.1 GROUP REPRESENTATION

This textual representation is displayed in Figure B.1. This textual representation is denoted as *Group Representation*, because it introduces additional *GROUP* elements in the text. These elements are used to group elements that are connected by the same logic connector. The elements belonging to one group are placed one level below the *GROUP* element. The curly brackets are used to make it easier to perceive, which elements belong together. Since the relations between the elements are already achieved by using different levels in the document, the brackets are optional. The groups themselves can be connected to other groups or elements via logical operators. In contrast to the *Original Representation*, the grouping avoids the repetition of elements and ensures that the propositional logic of the activity is correctly reproduced in the RS document. Another advantage is that the representation still resembles the structure of the activity diagram as the paths are still recognizable. Besides, its structure is closely related to the *Original Representation* and thus it is easy for the stakeholders to adapt to this new textual representation.

Additionally, a *THEN* operator is introduced that describes that *Actions* are executed consecutively. This means, that every *Action* only starts executing, when its predecessors have successfully finished their executions. This way, it is also possible to represent the order of executions of actions in the paths of the activity. An AND operator on the other hand represents a *JoinNode* and thus indicates that all connected elements can be independently executed. A drawback of this representation is that it needs additional grouping elements to correctly describe the activity diagrams' structure. As these elements are not requirements per se, the description becomes longer and also needs additional levels. This may impede the understandability of the function execution.

Text	Level	Type
<b>Drive Inhibit</b>	2	Function
GROUP {	3	-
GROUP {	4	-
Vehicle is in "P" AND	5	Trigger
GROUP {	5	-
State of connector "plugged" OR	6	Trigger
State of connector "vehicle_plugged" OR	6	Trigger
State of connector "unknown"	6	Trigger
}	5	-
} THEN	4	-
V < 5 km/h	4	Check
} OR	3	-
GROUP {	3	-
GROUP {	4	-
State of connector "plugged" OR	5	Trigger
State of connector "vehicle_plugged"	5	Trigger
} THEN	4	-
V < 5 km/h THEN	4	Check
Engine Cranking inactive	4	Check
} OR	3	-
GROUP {	3	-
GROUP {	4	-
State of connector "defect" OR	5	Trigger
State of connector "unknown"	5	Trigger
} THEN	4	-
Gearshift is in "P"	4	Check
}	3	-

Figure B.1: Group Representation

## B.2 NORMAL FORM REPRESENTATION

In Figure B.2 another alternative textual representation is displayed. This textual representation is called *Normal Form Representation* because it represents a disjunctive normal form of the propositional logic statement underlying the activity diagram. Therefore, this representation solely describes the aspect of propositional logic in activities and refrains from describing the execution order or parallel processing. In contrast to the *Original Representation*, this ensures that the propositional logic of the activity is correctly reproduced. Additionally, the conversion into a normal form simplifies the representation of the underlying propositional formula if possible.

Similar to the *Group Representation*, elements are structured into groups by inserting an object in the text, denoted with the string *GROUP*. All elements of a group are placed one level below that *GROUP* element. The elements of a group are logically connected by an AND operator (omitted in the example), while all the groups are connected by an OR operator.

Text	Level	Type
<b>Drive Inhibit</b>	2	Function
GROUP OR	3	-
State of connector "plugged"	4	Trigger
Vehicle is in "P"	4	Trigger
V < 5 km/h	4	Check
GROUP OR	3	-
State of connector "vehicle_plugged"	4	Trigger
Vehicle is in "P"	4	Trigger
V < 5 km/h	4	Check
GROUP OR	3	-
State of connector "unknown"	4	Trigger
Vehicle is in "P"	4	Trigger
V < 5 km/h	4	Check
GROUP OR	3	-
State of connector "vehicle_plugged"	4	Trigger
V < 5 km/h	4	Check
Engine Cranking inactive	4	Check
GROUP OR	3	-
State of connector "plugged"	4	Trigger
V < 5 km/h	4	Check
Engine Cranking inactive	4	Check
GROUP OR	3	-
State of connector "unknown"	4	Trigger
Gearshift is in "P"	4	Check
GROUP	3	-
State of connector "defect"	4	Trigger
Gearshift is in "P"	4	Check

Figure B.2: Normal Form Representation

Due to the OR connections between groups, an execution of all elements in any group results in the activation of the function. Therefore, this representation emphasizes distinct combinations of elements that cause a function's activation. In the normal form, only two hierarchical levels are needed to display the representation in the RS document.

One of the disadvantages of this representation is that its structure does not resemble the structure of the activity diagram. Additionally, the generation of the normal form suppresses the order of execution of the elements. Hence, the sequence in which the elements need to be executed, is not part of the representation. This drawback may be mitigated by using the order of appearance beneath a grouping element as an indicator for execution sequences. However, a group might also contain elements that are independently executable – an information that gets lost if order of appearance is interpreted as execution order. Additional structural elements would then be necessary to express the independence of certain elements.

### B.3 TREE REPRESENTATION

In Figure B.3 a third textual representation is displayed. This textual representation is denoted *Tree Representation* because it uses the hierarchical document structure to display the expression tree [92] of the propositional logic statements in the diagram. In this representation, the logic operators AND and OR are distinct objects in the RS document. All elements that are one level below an operator are logically connected by that operator. An operator element might have further operators one level below as elements. This ensures that logic relations between the elements of the diagram are correctly reproduced in the textual representation. The *Tree Representation* reflects the logical statement as it appears in the diagram (i.e., without simplifications or transformations).

Text	Level	Type
<b>Drive Inhibit</b>	2	Function
OR	3	-
AND	4	-
Vehicle is in "P"	5	Trigger
OR	5	-
State of connector "plugged"	6	Trigger
State of connector "vehicle_plugged"	6	Trigger
State of connector "unknown"	6	Trigger
V < 5 km/h	5	Check
AND	4	-
OR	5	-
State of connector "plugged"	6	Trigger
State of connector "vehicle_plugged"	6	Trigger
V < 5 km/h	5	Check
Engine Cranking inactive	5	Check
AND	4	-
OR	5	-
State of connector "defect"	6	Trigger
State of connector "unknown"	6	Trigger
Gearshift is in "P"	5	Check

Figure B.3: Tree Representation

This representation has the drawback that its structure does not resemble the structure of the original activity diagram, which makes it harder to recognize the relations between the two artifacts. Also, a basic understanding of expression trees may be necessary to comprehend the connections between the elements. As tree structures are suitable to express all types of formulas, it is also possible to add more operators aside from the propositional logic operators (e.g., including a *THEN* operator for expressing sequences of actions).



## B.4 EXACT EQUIVALENT REPRESENTATION

This textual representation of activity diagrams was suggested by Flater et al. [84]. Their idea is to convert the complete activity diagram into human-readable text. We call this *Exact Equivalent Representation* because it maps each visual element to a textual string. Thus the resulting text includes every aspect of the original activity diagram and is therefore applicable to all sorts of activity diagrams. Elements of activity diagrams are represented by symbols such as parenthesis (as actions nodes) and square brackets (as object nodes). *ActivityEdges* are depicted by ASCII arrows (<- and ->). Elements used multiple times contain the name of the elements followed by an asterisk and a variable name, which is used as reference. Since the referenced work did not include *AcceptEventActions*, we use a greater-than sign and a square bracket (> *ElementText* ]) for these elements.

Text	Level	Type
<b>Drive Inhibit</b>	2	Function
>Trigger: Vehicle is in "P"] -> <JoinNode *jn1> -> (Check: V < 5 km/h) -> <MergeNode *mn1> -> (Function: Drive Inhibit) -> <ActivityFinal> >Trigger: State of connector "vehicle_plugged"] -> <MergeNode *mn2> -> <*jn1> >Trigger: State of connector "plugged"] -> <*mn2> >Trigger: State of connector "unknown"] -> <*mn2>  >Trigger: State of connector "vehicle_plugged"] -> <MergeNode *mn3> -> (Check: V < 5 km/h) -> (Check: Engine Cranking inactive) -> <*mn1> >Trigger: State of connector "plugged"] -> <MergeNode *mn3>  >Trigger: State of connector "defect"] -> <MergeNode *mn4> -> (Check: Gearshift is in "P") -> <*mn1> >Trigger: State of connector "unknown"] -> <MergeNode *mn4>	3	-

Figure B.4: Exact Equivalent Representation

The resulting textual representation is shown in Figure B.4. Since the textual notation is a serialization of the activity, the multiple levels of a hierarchically structured document are not necessary and the transformation is placed in a single entry. This representation has no loss of information besides the layout of the activity diagram. The authors mention that using this textual representation instead of graphical representations does not require special tooling and reduces effort when implementing prototypes [84]. However, not using special tooling allows for the construction of invalid expressions.

A major drawback of this representation is that it is difficult to get a quick overview of the function and to grasp the relations between elements. In addition, stakeholders not familiar with activity diagrams have no advantage in understanding this type of text better than an activity diagram itself.

## B.5 EVALUATION OF THE TEXTUAL REPRESENTATIONS

To assess how practitioners perceive the usefulness of the presented representations and to learn more about their preferences, a survey is carried out in which the stakeholders of the examined system in this thesis are asked to create a ranking of the representations.

The survey consists of three parts. First, the survey presents an original activity diagram of a function of the system the participants are involved in. The function is the same as the sample in Chapter 3 and Chapter 4. However, the presented activity diagram looks as in the tool (Enterprise Architect) used by the participants. The one in their tool and the one displayed in this thesis differ slightly in terms of color and layout.

Then, the activity diagram and its different textual representations are presented. Again, the textual representations are presented as they would look in the tool (IBM DOORS) used by the participants. Each textual representation is accompanied by an explanatory text and a listing of its advantages and disadvantages.

Finally, the survey document contains a pairwise comparison of the textual representations. Since we examine five different textual representations, the pairwise comparison consists of ten comparisons to include all combinations. For each comparison, the participants had to provide which representation they perceive as more useful or whether both are equally useful. Each time a representation surpasses another, it is accredited with 1 point, while the other representation is accredited with -1 point. In case of a tie, both representations receive 0 points. For the ranking the points are added together. Besides, the participants had the chance to leave comments as a freely written text. Six stakeholders participated in the survey.

### B.5.1 Results

The rankings of the individual participants are shown in Table B.1. Its entries are sorted by descending order of the participant's preferences. If two representations are not separated by a horizontal line, the representations were ranked equally. Next to the name of the representation, in the brackets, are the decisions the participants made for the representation. The first number stands for how often it was preferred, the second number for the number of ties and the last number for how often it was considered less useful.

Table B.2 shows the aggregated results for all participants. The first column shows the aggregated ranking by combining all the decisions taken by the participants. The number in the brackets next to the representation is the result of the computation mentioned for the individual participants (1 point - better, 0 points - tie, -1 point - worse), applied to all decisions. The second column shows an aggregation based on an assignment of points for the individual ranks: A textual

Table B.1: Ranking of textual representations for each participant in descending order

Participant 1	Participant 2	Participant 3	Participant 4	Participant 5	Participant 6
Normal Form (4:0:0)	Group (4:0:0)	Group (4:0:0)	Group (3:1:0)	Original (4:0:0)	Tree (4:0:0)
Group (3:0:1)	Tree (3:0:1)	Original (3:0:1)	Tree (3:1:0)	Tree (3:0:1)	Group (3:1:0)
Original (1:1:2)	Original (1:1:2)	Tree (1:1:2)	Original (2:0:2)	Group (2:0:2)	Normal Form (2:0:2)
Tree (1:1:2)	Normal Form (0:2:2)	Normal Form (1:0:3)	Normal Form (0:1:3)	Normal Form (1:0:3)	Exact Equivalent (0:1:3)
Exact Equivalent (0:0:4)	Exact Equivalent (0:1:3)	Exact Equivalent (0:1:3)	Exact Equivalent (0:1:3)	Exact Equivalent (0:0:4)	Original (0:0:4)

Table B.2: Aggregated ranking of textual representations for all participants

Rank	Combined	Points
1	Group (15)	Group (25.5)
2	Tree (9)	Tree (23)
3	Original (0)	Original (18.5)
4	Normal Form (-5)	Normal Form (15.5)
5	Exact Equivalent (-19)	Exact Equivalent (7.5)

representation on the first rank receives five points, while the one on the last rank receives one. In case both have the same rank, the points of the respective ranks are added together and divided by the number of representations of the same rank. For example, for *Participant 1*, the representation on the third rank receives three points, on the fourth rank it receives two points. As there are two representations on the same rank both receive 2.5 points (the average). The resulting points are shown in the brackets next to the name of the representation.

The resulting rankings for both aggregations are the same. Nonetheless, the separations of the representations differ between the aggregations. While the ranking by points is close together, the representations in the combined ranking are farther apart. Especially, the *Group Representation* is far ahead in the combined aggregation because it was preferred in most pairwise decisions.

Besides the pairwise comparisons, one participant used the opportunity to leave a comment. *Participant 2* stated the *Group Representation* improves readability since confusing repetitions of elements are avoided.

### B.5.2 Discussion

Based on the ranking the *Group Representation* achieved in both the individual and the aggregated rankings, it can be considered as the most appropriate representation for the participants. The high acceptance of this representation may have resulted from the fact that it resembles the structure of the currently used (original) representation and is thus familiar to the participants. Furthermore, it mitigates some of the weaknesses of the *Original Representation* such as non-atomic entries and unclear logic relations between the entries. Due to the similarity with the *Original Representation*, it is easy for participants to comprehend the *Group Representation*. As such, the *Group Representation* was most likely perceived as an improved version of the representation currently used.

The *Exact Equivalent Representation* was ranked the least preferable. Hence, a mere transformation of an activity diagram into text is not considered adequate by the participants. This is probably caused by the reduced readability, which makes the function harder to understand. This is also in accordance with the fact that graphical models are used to improve understandability in the first place.

Both the *Normal Form Representation* as well as the *Tree Representation* achieved good rankings for individual participants. As a consequence, as long as all representations are kept consistent with each other, the representations could exist side by side as views of the same function. This approach also has the advantage that implicit information in the models can be made explicit depending on the individual needs and the background of each stakeholder.

## TOOL PROTOTYPE: AUTOMATIC GENERATION OF TEXTUAL REPRESENTATIONS FROM UML<sub>2</sub> ACTIVITY DIAGRAMS

One of the suggestions in Chapter 3 is to automatically generate the textual representations from the activity diagrams in order to avoid inconsistencies between the coexisting representations and other quality issues. This is made possible by the contributions presented in Chapter 5 and Chapter 6. In the following a tool prototype is presented that implements these concepts and automates the process. First, the structure of the tool is demonstrated followed by an overview of its functionality.

### C.1 STRUCTURE

Although the industry partner uses IBM DOORS and Enterprise Architect as its tools for requirements engineering and for modeling tasks, the prototype is designed to account for possible changes of the employed tools. The resulting (automatic) procedure is shown in Figure C.1.

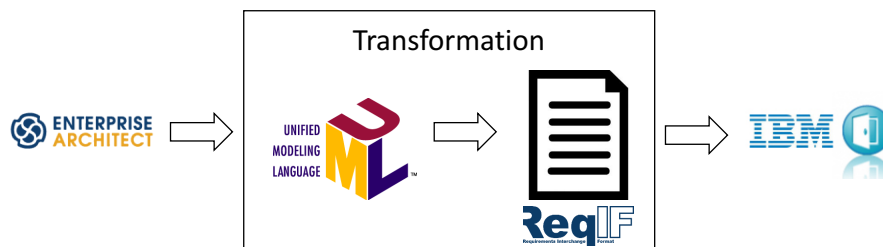


Figure C.1: Procedure of the text generation from activity diagrams

To enable the support of different requirements engineering and modeling tools, the tools are integrated by the prototype with adapters. These adapters ensure that the transformation from the activity diagram to the textual representation is realized in a tool-independent manner. The adapters transform the tool-dependent formats into the internally used tool-independent formats. Concretely, for the activity diagram an EMF format [71] is used that represents the activity diagrams precisely as defined by the UML<sub>2</sub> specification.

The procedure is similar for the textual representation. As a tool-independent solution ReqIF [173] is used. Since most RE tools already support the import of ReqIF files, the transformation to a tool-specific meta-model might not be necessary. However, to improve usability, an

adapter to IBM DOORS is implemented nonetheless, to facilitate the placement of the textual parts within an [RS](#) document.

Overall, this concept enables a tool-independent generation of the textual representation from the [UML2](#) activity diagrams. In case one of the tools is replaced by another, only the corresponding adapter needs to be replaced.

## C.2 OVERVIEW

The basic functionality of the tool prototype is depicted in the screenshots in [Figure C.2](#).

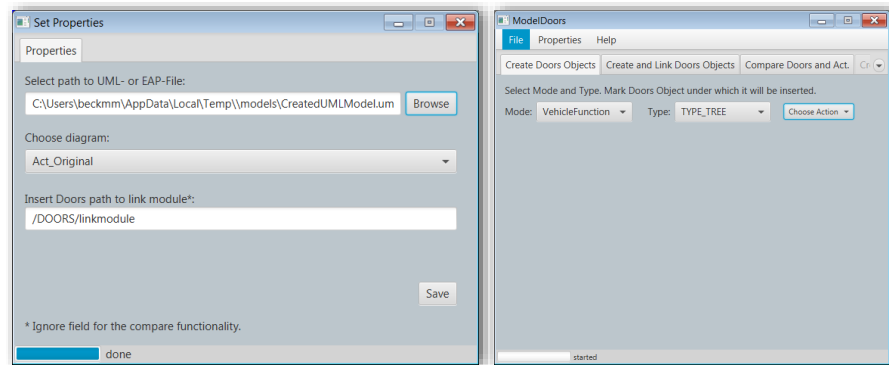


Figure C.2: Dialogs of the tool prototype

The dialog on the left hand side is used to specify the Enterprise Architect project, to select an activity diagram, and to choose the *Link Module* of IBM DOORS. The latter is needed to establish the traceability links between the model elements and the objects in IBM DOORS. It is necessary to have traceability information available in both representations to ensure bidirectional traceability. Because this, in turn, is demanded by development process standards.

The dialog on the right hand side is used to select the type of textual representation that will be generated (see [Appendix B](#) for the possibilities). The textual representation will be generated below the marked object (one level below in the hierarchy level) in IBM DOORS. The tool prototype also features additional functionality like updating existing textual representations (that have been automatically generated before) and indicating differences between the representations.

## BIBLIOGRAPHY

---

- [1] Han van der Aa, Henrik Leopold, Felix Mannhardt, and Hajo A. Reijers. "On the Fragmentation of Process Information: Challenges, Solutions, and Outlook." In: *International Conference on Enterprise, Business-Process and Information Systems Modeling* (2015).
- [2] Han van der Aa, Henrik Leopold, and Hajo A. Reijers. "Comparing textual descriptions to process models – The automatic detection of inconsistencies." In: *Information Systems* 64 (2017).
- [3] Han van der Aa, Henrik Leopold, Inge van de Weerd, and Hajo A. Reijers. "Causes and Consequences of Fragmented Process Information: Insights from a Case Study." In: *Americas Conference on Information Systems* (2017).
- [4] Russell J. Abbott. "Program Design by Informal English Descriptions." In: *Communications of the ACM* 26.11 (1983).
- [5] Mariem Abdouli, Wahiba Ben Abdesslem Karaa, and Henda Ben Ghezala. "Survey of Works that Transform Requirements into UML Diagrams." In: *International Conference on Software Engineering Research, Management and Applications* (2016).
- [6] Radio Technical Commission for Aeronautics Inc. (RTCA). *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*. 1992.
- [7] Mudassar Adeel Ahmed, Wasi Haider Butt, Imran Ahsan, Muhammad Waseem Anwar, Muhammad Latif, and Farooque Azam. "A Novel Natural Language Processing (NLP) Approach to Automatically Generate Conceptual Class Model from Initial Software Requirements." In: *International Conference on Information Science and Applications* (2017).
- [8] Thomas Allweyer. *BPMN 2.0: Introduction to the Standard for Business Process Modeling*. BoD–Books on Demand, 2016.
- [9] Dalal Alrajeh, Alessandra Russo, and Sebastian Uchitel. "Inferring Operational Requirements from Scenarios and Goal Models Using Inductive Learning." In: *International Workshop on Scenarios and State machines: Models, Algorithms, and Tools* (2006).
- [10] Talat Ambreen, Naveed Ikram, Muhammad Usman, and Mahmood Niazi. "Empirical research in requirements engineering: trends and opportunities." In: *Requirements Engineering* 23.1 (2018).

- [11] Vincenzo Ambriola and Vincenzo Gervasi. "On the Systematic Analysis of Natural Language Requirements with CIRCE." In: *Automated Software Engineering* 13.1 (2006).
- [12] Annie I. Antón and Colin Potts. "The Use of Goals to sSurface Requirements for Evolving Systems." In: *International Conference on Software Engineering* (1998).
- [13] Paul Arkley and Steve Riddle. "Overcoming the Traceability Benefit Problem." In: *International Conference on Requirements Engineering* (2005).
- [14] Jim Arlow, Wolfgang Emmerich, and John Quinn. "Literate Modelling — Capturing Business Knowledge with the UML." In: *International Conference on the Unified Modeling Language* (1998).
- [15] Bruno M. Arnaut, Denise B. Ferrari, and Marcelo Lopes de Oliveira e Souza. "A Requirements Engineering and Management Process in Concept Phase of Complex Systems." In: *International Symposium on Systems Engineering* (2016).
- [16] Saïd Assar. "Model Driven Requirements Engineering: Mapping the Field and Beyond." In: *International Model-Driven Requirements Engineering Workshop* (2014).
- [17] Colin Atkinson and Thomas Kuhne. "Model-Driven Development: a Metamodeling Foundation." In: *IEEE Software* 20.5 (2003).
- [18] Banu Aysolmaz, Mehmet Gürsul, Kathrin Kirchner, Ralf Laue, Robert Mertens, Felix Reher, Irene M Schönreiter, Bernhard M. Turban, and Rüdiger Weißbach. "A Reflection on the Interrelations Between Business Process Management and Requirements Engineering with an Agility Perspective." In: *International Conference on Business Process Management* (2017).
- [19] Banu Aysolmaz, Henrik Leopold, Hajo A. Reijers, and Onur Demirörs. "A semi-automated approach for generating natural language requirements documents based on business process models." In: *Information and Software Technology* 93 (2018).
- [20] Marko Bajec, Damjan Vavpotič, and Marjan Krisper. "Practice-driven approach for creating project-specific software development methods." In: *Information and Software technology* 49.4 (2007).
- [21] Martin Beckmann, Thomas Karbe, and Andreas Vogelsang. "Information Extraction from High-Level Activity Diagrams to Support Development Tasks." In: *International Conference on Model-Driven Engineering and Software Development* (2018).
- [22] Martin Beckmann, Vanessa N. Michalke, Andreas Vogelsang, and Aaron Schlutter. "Removal of Redundant Elements within UML Activity Diagrams." In: *International Conference on Model Driven Engineering Languages and Systems* (2017).



- [23] Martin Beckmann, Christian Reuter, and Andreas Vogelsang. "Coexisting Graphical and Structured-Textual Representations of Requirements: Insights and Suggestions." In: *International Working Conference on Requirements Engineering: Foundation for Software Quality* (2018).
- [24] Martin Beckmann and Andreas Vogelsang. "What is a Good Textual Representation of Activity Diagrams in Requirements Documents?" In: *International Model-Driven Requirements Engineering Workshop* (2017).
- [25] Martin Beckmann, Andreas Vogelsang, and Christian Reuter. "A Case Study on a Specification Approach using Activity Diagrams in Requirements Documents." In: *International Requirements Engineering Conference* (2017).
- [26] Brian Berenbach. "The Automated Extraction of Requirements from UML Models." In: *International Requirements Engineering Conference* (2003).
- [27] Brian Berenbach. "Comparison of UML and Text based Requirements Engineering." In: *Conference on Object-Oriented Programming Systems, Languages, and Applications* (2004).
- [28] Brian Berenbach. "A 25 Year Retrospective on Model-Driven Requirements Engineering." In: *Model-Driven Requirements Engineering Workshop* (2012).
- [29] Daniel M. Berry, Ricardo Gacitua, Pete Sawyer, and Sri Fatimah Tjong. "The Case for Dumb Requirements Engineering Tools." In: *International Working Conference on Requirements Engineering: Foundation for Software Quality* (2012).
- [30] Daniel M. Berry, Erik Kamsties, and Michael M. Krieger. *From contract drafting to software specification: linguistic sources of ambiguity*. Tech. rep. University of Waterloo, 2003.
- [31] Robert Blumberg and Shaku Atre. "The Problem with Unstructured Data." In: *DM Review* 13 (2003).
- [32] International Requirements Engineering Board. "A Glossary of Requirements Engineering Terminology." In: *Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam* (2014).
- [33] Conrad Bock. "UML 2 Activity and Action Models." In: *Journal of Object Technology* 2.5 (2003).
- [34] Conrad Bock. "UML without Pictures." In: *IEEE Software* 20.5 (2003).
- [35] Narasimha Bolloju and Sherry X. Y. Sun. "Benefits of supplementing use case narratives with activity diagrams — An exploratory study." In: *Journal of Systems and Software* 85.9 (2012).

- [36] Grady Booch, Alan W. Brown, Sridhar Iyengar, James Rumbaugh, and Bran Selic. "An MDA Manifesto." In: *MDA Journal* 5.2 (2004).
- [37] Peter Braun, Manfred Broy, Frank Houdek, Matthias Kirchmayr, Mark Müller, Birgit Penzenstadler, Klaus Pohl, and Thorsten Weyer. "Guiding requirements engineering for software-intensive embedded systems in the automotive industry." In: *Computer Science-Research and Development* 29.1 (2014).
- [38] Lionel Briand and Yvan Labiche. "A UML-Based Approach to System Testing." In: *Software and Systems Modeling* 1.1 (2002).
- [39] Manfred Broy. "Automotive Software and Systems Engineering." In: *International Conference on Formal Methods and Models for Co-Design* (2005).
- [40] Manfred Broy. "Challenges in Automotive Software Engineering." In: *International Conference on Software Engineering* (2006).
- [41] Manfred Broy, Werner Damm, Stefan Henkler, Klaus Pohl, Andreas Vogelsang, and Thorsten Weyer. "Introduction to the SPES modeling framework." In: *Model-Based Engineering of Embedded Systems*. Springer, 2012.
- [42] Manfred Broy, Ingolf H. Kruger, Alexander Pretschner, and Christian Salzmann. "Engineering Automotive Software." In: *Proceedings of the IEEE* 95.2 (2007).
- [43] Håkan Burden and Rogardt Heldal. "Natural Language Generation from Class Diagrams." In: *International Workshop on Model-Driven Engineering, Verification and Validation* (2011).
- [44] Andrew Burton-Jones and Peter N. Meso. "The Effects of Decomposition Quality and Multiple Forms of Information on Novices' Understanding of a Domain from a Conceptual Model." In: *Journal of the Association for Information Systems* 9.12 (2008).
- [45] Gustavo Cabral and Augusto Sampaio. "Formal Specification Generation from Requirement Documents." In: *Electronic Notes in Theoretical Computer Science* 195 (2008).
- [46] A. Moreno Capuchino, Natalia Juristo, and Reind P. Van de Riet. "Formal justification in object-oriented modelling: A linguistic approach." In: *Data & Knowledge Engineering* 33.1 (2000).
- [47] Evellin Cristine Souza Cardoso, João Paulo A. Almeida, and Giancarlo Guizzardi. "Requirements Engineering Based on Business Process Models A Case Study." In: *Enterprise Distributed Object Computing Conference Workshops* (2009).
- [48] Carla Carnaghan. "Business process modeling approaches in the context of process level audit risk assessment: An analysis and comparison." In: *International Journal of Accounting Information Systems* 7.2 (2006).

- [49] Jane Cleland-Huang, Orlena Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. "Software Traceability: Trends and Future Directions." In: *Future of Software Engineering* (2014).
- [50] Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman. *Software and Systems Traceability*. Vol. 2. 3. Springer, 2012.
- [51] CMMI Product Team. *CMMI for Development, Version 1.3*. Pittsburgh, PA, 2010. URL: <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9661>.
- [52] Ahmet Coşkunçay, Banu Aysolmaz, Onur Demirörs, Omer Bilen, and Idris Dogani. "Bridging the Gap between Business Process Modeling and Software Requirements Analysis: A Case Study." In: *Mediterranean Conference on Information Systems* (2010).
- [53] Karl Cox, Keith T. Phalp, Steven J. Bleistein, and June M. Verner. "Deriving requirements from process models via the problem frames approach." In: *Information and Software Technology* 47:5 (2005).
- [54] Krzysztof Czarnecki and Simon Helsen. "Feature-based survey of model transformation approaches." In: *IBM Systems Journal* 45:3 (2006).
- [55] Jesse Daniels and Terry Bahill. "The Hybrid Process That Combines Traditional Requirements and Use Cases." In: *Systems Engineering* 7:4 (2004).
- [56] Marian Daun, Bastian Tenbergen, and Thorsten Weyer. "Requirements viewpoint." In: *Model-Based Engineering of Embedded Systems*. Springer, 2012.
- [57] Islay Davies, Peter Green, Michael Rosemann, Marta Indulska, and Stan Gallo. "How do practitioners use conceptual modeling in practice?" In: *Data & Knowledge Engineering* 58:3 (2006).
- [58] Zamira Daw and Rance Cleaveland. "Comparing model checkers for timed UML activity diagrams." In: *Science of Computer Programming* 111 (2015).
- [59] 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." In: *Information and Software Technology* 54:10 (2012).
- [60] Jose Luis De la Vara González and J. Sanchez Díaz. "Business process-driven requirements engineering: A goal-based approach." In: *Workshop on Business Process Modeling* (2007).
- [61] Renaud De Landtsheer, Emmanuel Letier, and Axel van Lamswerde. "Deriving tabular event-based specifications from goal-oriented requirements models." In: *Requirements Engineering* 9:2 (2004).

- [62] Fabio Cardoso De Souza and Fernando Antonio de Castro Giorno. "Automatic Generation of Sequence Diagrams and Updating Domain Model from Use Cases." In: *International Conference on Advances and Trends in Software Engineering* (2015).
- [63] Narayan Debnath, María Carmen Leonardi, María Virginia Mauco, Germán Montejano, and Daniel Riesco. "Improving Model Driven Architecture with Requirements Models." In: *International Conference on Information Technology: New Generations* (2008).
- [64] Christian Denger, Daniel M. Berry, and Erik Kamsties. "Higher Quality Requirements Specifications through Natural Language Patterns." In: *International Conference on Software: Science, Technology and Engineering* (2003).
- [65] Jeremy Dick, Elizabeth Hull, and Ken Jackson. *Requirements Engineering*. Vol. 3. Springer, 2017.
- [66] Remco Dijkman, Marlon Dumas, Boudewijn Van Dongen, Reina Käärik, and Jan Mendling. "Similarity of business process models: Metrics and evaluation." In: *Information Systems* 36.2 (2011).
- [67] Ahmet Dikici, Oktay Türetken, and Onur Demirörs. "Factors influencing the understandability of process models: A systematic literature review." In: *Information and Software Technology* (2017).
- [68] Brian Dobing and Jeffrey Parsons. "How UML is used." In: *Communications of the ACM* 49.5 (2006).
- [69] Doron Drusinsky. "From UML Activity Diagrams to Specification Requirements." In: *International Conference on System of Systems Engineering* (2008).
- [70] Eclipse Foundation, Inc. *ProR Requirements Engineering Platform*. (<https://www.eclipse.org/rmf/pror/>). Accessed: 2018-03-30.
- [71] Eclipse Modeling Framework (EMF). (<http://www.eclipse.org/modeling/emf/>). Accessed: 2018-06-05.
- [72] Martin Eigner, Walter Koch, and Christian Muggeo. *Modellbasierter Entwicklungsprozess cybertronischer Systeme*. Vol. 1. Springer, 2017.
- [73] Meryem Elallaoui, Khalid Nafil, and Raja Touahni. "Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques." In: *International Conference on Ambient Systems, Networks and Technologies* 130 (2018).
- [74] Elena Viorica Epure, Patricia Martín-Rodilla, Charlotte Hug, Rebecca Deneckère, and Camille Salinesi. "Automatic Process Model Discovery from Textual Methodologies." In: *International Conference on Research Challenges in Information Science* (2015).

- [75] Henrik Eriksson. "The semantic-document approach to combining documents and ontologies." In: *International Journal of Human-Computer Studies* 65.7 (2007).
- [76] Rik Eshuis and Roel Wieringa. "Tool Support for Verifying UML Activity Diagrams." In: *Transactions on Software Engineering* 30.7 (2004).
- [77] John Favaro, Silvia Mazzini, Rudolf Schreiner, Hans-Peter König, and Xavier Olive. "Next Generation Requirements Engineering." In: *INCOSE International Symposium* 22.1 (2014).
- [78] Loe M. G. Feijs. "Natural language and message sequence chart representation of use cases." In: *Information and Software Technology* 42.9 (2000).
- [79] Henning Femmer and Andreas Vogelsang. "Requirements Quality is Quality in Use." In: *IEEE Software* (2018).
- [80] Alessio Ferrari, Giorgio O. Spagnolo, and Felice Dell'Orletta. "Mining Commonalities and Variabilities from Natural Language Documents." In: *International Software Product Line Conference* (2013).
- [81] Anthony Finkelstein and Wolfgang Emmerich. "The Future of Requirements Management Tools." In: *Information Systems in Public Administration and Law*. (2000).
- [82] Donald Firesmith. "Modern Requirements Specification." In: *Journal of Object Technology* 2.2 (2003).
- [83] Donald Firesmith. "Generating Complete, Unambiguous, and Verifiable Requirements from Stories, Scenarios, and Use Cases." In: *Journal of Object Technology* 3 (Nov. 2004).
- [84] David Flater, Philippe Martin, and Michelle Crane. *Rendering UML Activity Diagrams as Human-Readable Text*. Tech. rep. NISTIR 7469, National Institute of Standards and Technology, 2009.
- [85] Günther Fliedl, Christian Kop, Heinrich C. Mayr, Alexander Salbrechter, Jürgen Vöhringer, Georg Weber, and Christian Winkler. "Deriving static and dynamic concepts from software requirements using sophisticated tagging." In: *Data & Knowledge Engineering* 61.3 (2007).
- [86] Markus Fockel and Jörg Holtmann. "A Requirements Engineering Methodology Combining Models and Controlled Natural Language." In: *International Model-Driven Requirements Engineering Workshop* (2014).
- [87] Martin Fowler. *UML distilled*. 3rd ed. Addison-Wesley, 2003.
- [88] Robert France and Bernhard Rumpe. "Model-Driven Development of Complex Software: A Research Roadmap." In: *2007 Future of Software Engineering* (2007).

- [89] Fabian Friedrich, Jan Mendling, and Frank Puhlmann. "Process Model Generation from Natural Language Text." In: *International Conference on Advanced Information Systems Engineering* (2011).
- [90] Dariusz Gall and Anita Walkowiak. "An Approach to Semantics for UML Activities." In: *International Conference on Information Systems Architecture and Technology* (2017).
- [91] Andrew Gemino. "Empirical comparisons of animation and narration in requirements validation." In: *Requirements Engineering* 9.3 (2004).
- [92] Richard Gilberg and Behrouz Forouzan. *Data Structures: A pseudocode approach with C*. Nelson Education, 2004.
- [93] Michal Gordon and David Harel. "Steps Towards Scenario-Based Programming with a Natural Language Interface." In: *Joint European Conferences on Theory and Practice of Software* (2014).
- [94] Tony Gorschek, Per Garre, Stig Larsson, and Claes Wohlin. "A Model for Technology Transfer in Practice." In: *IEEE software* 23.6 (2006).
- [95] Orlena Gotel, Jane Cleland-Huang, Jane H. Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, and Giuliano Antoniol. "The Quest for Ubiquity: A Roadmap for Software and Systems Traceability Research." In: *International Requirements Engineering Conference* (2012).
- [96] Orlena Gotel and C.W. Finkelstein. "An Analysis of the Requirements Traceability Problem." In: *International Conference on Requirements Engineering* (1994).
- [97] Klaus Grimm. "Software Technology in an Automotive Company: Major Challenges." In: *International Conference on Software Engineering* (2003).
- [98] Hans Grönniger, Dirk Reiß, and Bernhard Rumpe. "Towards a Semantics of Activity Diagrams with Semantic Variation Points." In: *International Conference on Model Driven Engineering Languages and Systems* (2010).
- [99] Anne Gross and Joerg Doerr. "EPC vs. UML Activity Diagram - Two Experiments Examining their Usefulness for Requirements Engineering." In: *International Requirements Engineering Conference* (2009).
- [100] Anne Gross and Joerg Doerr. "What You Need Is What You Get!: The Vision of View-Based Requirements Specifications." In: *International Requirements Engineering Conference* (2012).
- [101] Paul Grünbacher, Alexander Egyed, and Nenad Medvidovic. "Reconciling software requirements and architectures with intermediate models." In: *Software & Systems Modeling* 3.3 (2004).

- [102] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. "Semantically Enhanced Software Traceability Using Deep Learning Techniques." In: *International Conference on Software Engineering* (2017).
- [103] Alireza Haghighatkhah, Ahmad Banijamali, Olli-Pekka Pakenen, Markku Oivo, and Pasi Kuvaja. "Automotive software engineering: A systematic mapping study." In: *Journal of Systems and Software* 128 (2017).
- [104] Terry Halpin. "Object-Role Modeling (ORM/NIAM)." In: *Handbook on Architectures of Information Systems* (1998).
- [105] Sean Hansen, Nicholas Berente, and Kalle Lyytinen. "Requirements in the 21st Century: Current Practice and Emerging Trends." In: *Design requirements engineering: A ten-year perspective* (2009).
- [106] David Harel and Bernhard Rumpe. *Modeling Languages Syntax, Semantics and all that Stuff (or, What's the Semantics of "Semantics")*. Tech. rep. 2004.
- [107] Jeffrey Heer and Maneesh Agrawala. "Software Design Patterns for Information Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006).
- [108] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. "Design Science in Information Systems Research." In: *MIS quarterly* 28.1 (2004).
- [109] Hubert F. Hofmann and Franz Lehner. "Requirements Engineering as a Success Factor in Software Projects." In: *IEEE Software* 18.4 (2001).
- [110] Jan Holmström, Mikko Ketokivi, and Ari-Pekka Hameri. "Bridging Practice and Theory: A Design Science Approach." In: *Decision Sciences* 40.1 (2009).
- [111] Jon Holt and Simon Perry. *Sysml for Systems Engineering-a Model-based Approach*. 2nd ed. Institution of Engineering and Technology, 2013.
- [112] Krzysztof Honkisz, Krzysztof Kluza, and Piotr Wiśniewski. "A Concept for Generating Business Process Models from Natural Language Description." In: *International Conference on Knowledge Science, Engineering and Management* (2018).
- [113] John Hutchinson, Mark Rouncefield, and Jon Whittle. "Model-driven Engineering Practices in Industry." In: *International Conference on Software Engineering* (2011).
- [114] John Hutchinson, Jon Whittle, and Mark Rouncefield. "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure." In: *Science of Computer Programming* 89 (2014).

- [115] IBM Corporation. *Rational DOORS*. (<https://www.ibm.com/us-en/marketplace/rational-doors>). Accessed: 2018-03-13.
- [116] M.G. Ilieva and Olga Ormandjieva. "Models Derived from Automatically Analyzed Textual User Requirements." In: *International Conference on Software Engineering Research, Management and Applications* (2006).
- [117] Emilio Insfrán, Oscar Pastor, and Roel Wieringa. "Requirements Engineering-Based Conceptual Modelling." In: *Requirements Engineering* 7.2 (2002).
- [118] International Electrotechnical Commission. *IEC 60617 - Graphical Symbols for Diagrams*. 1996.
- [119] International Electrotechnical Commission. *ISO / IEC TR 24766: 2009, Information technology – Systems and software engineering – Guide for requirements engineering tool capabilities*. 2009.
- [120] International Organization for Standardization. *ISO 128 Technical Drawings*. 2014.
- [121] Diaz Isabel, Oscar Pastor, and Alfredo Matteo. "Modeling interactions using role-driven patterns." In: *International Conference on Requirements Engineering* (2005).
- [122] Nili Itzik and Iris Reinhartz-Berger. "Generating Feature Models from Requirements: Structural vs. Functional Perspectives." In: *International Software Product Line Conference* (2014).
- [123] Martin Ivarsson, Fredrik Pettersson, and Peter Öhman. "Improved Control of Automotive Software Suppliers." In: *International Conference on Product Focused Software Process Improvement* (2005).
- [124] Stefan Jungmayr and Jens Stumpe. "Another Motivation for Usage Models: Generation of User Documentation." In: *CONQUEST* 98 (1998).
- [125] Mohamad Kassab. "The Changing Landscape of Requirements Engineering Practices over the Past Decade." In: *International Workshop on Empirical Requirements Engineering* (2015).
- [126] Mohamad Kassab, Colin Neill, and Phillip Laplante. "State of practice in requirements engineering: contemporary data." In: *Innovations in Systems and Software Engineering* 10.4 (2014).
- [127] Donald Ervin Knuth. "Literate programming." In: *The Computer Journal* 27.2 (1984).
- [128] Leonid Kof. "From Textual Scenarios to Message Sequence Charts Inclusion of Condition Generation and Actor Extraction." In: *International Requirements Engineering Conference* (2008).



- [129] Leonid Kof. "Translation of Textual Specifications to Automata by Means of Discourse Context Modeling." In: *International Working Conference on Requirements Engineering: Foundation for Software Quality* (2009).
- [130] Axel van Lamsweerde. "Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice." In: *International Requirements Engineering Conference* (2004).
- [131] Axel van Lamsweerde and Laurent Willemet. "Inferring Declarative Requirements Specifications from Operational Scenarios." In: *Software Engineering* 24.12 (1998).
- [132] Kevin Lano. *UML 2 Semantics and Applications*. John Wiley & Sons, 2009.
- [133] Henrik Leopold, Jan Mendling, and Artem Polyvyanyy. "Supporting Process Model Validation through Natural Language Generation." In: *Transactions on Software Engineering* 40.8 (2014).
- [134] Henrik Leopold, Fabian Pittke, and Jan Mendling. "Ensuring the canonicity of process models." In: *Data & Knowledge Engineering* 111 (2017).
- [135] Emmanuel Letier and Axel van Lamsweerde. "Deriving Operational Software Specifications from System Goals." In: *ACM SIGSOFT Software Engineering Notes* 27.6 (2002).
- [136] Keletso J. Letsholo, Liping Zhao, and Erol-Valeriu Chioasca. "TRAM: A Tool for Transforming Textual Requirements into Analysis Models." In: *International Conference on Automated Software Engineering* (2013).
- [137] Juan Li, Ross Jeffery, Kam Hay Fung, Liming Zhu, Qing Wang, He Zhang, and Xiwei Xu. "A Business Process-Driven Approach for Requirements Dependency Analysis." In: *International Conference on Business Process Management* (2012).
- [138] Grzegorz Loniewski, Emilio Insfrán, and Silvia Abrahão. "A Systematic Review of the Use of Requirements Engineering Techniques in Model-Driven Development." In: *International Conference on Model Driven Engineering Languages and Systems* (2010).
- [139] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E.M. van der Werf, and Sjaak Brinkkemper. "The Use and Effectiveness of User Stories in Practice." In: *International Working Conference on Requirements Engineering: Foundation for Software Quality* (2016).
- [140] Garm Lucassen, Marcel Robeer, Fabiano Dalpiaz, Jan Martijn E.M. van der Werf, and Sjaak Brinkkemper. "Extracting conceptual models from user stories with Visual Narrator." In: *Requirements Engineering* 22.3 (2017).

- [141] Neil A. M. Maiden, Sharon Manning, Sara Jones, and John Greenwood. "Generating requirements from systems models using patterns: a case study." In: *Requirements Engineering* 10.4 (2005).
- [142] Neil A.M. Maiden, Shailey Minocha, Keith Manning, and Michele Ryan. "CREWS-SAVRE: Systematic Scenario Generation and Use." In: *International Conference on Requirements Engineering* (1998).
- [143] Neil Maiden and Suzanne Robertson. "Developing Use Cases and Scenarios in the Requirements Process." In: *International Conference on Software Engineering* (2005).
- [144] Saleem Malik and Imran Sarwar Bajwa. "Back to Origin: Transformation of Business Process Models to Business Rules." In: *International Conference on Business Process Management* (2012).
- [145] Bilal Maqbool, Farooque Azam, Muhammad Waseem Anwar, Wasi Haider Butt, Jahan Zeb, Iqra Zafar, Aiman Khan Nazir, and Zuneera Umair. "A Comprehensive Investigation of BPMN Models Generation from Textual Requirements—Techniques, Tools and Trends." In: *International Conference on Information Science and Applications* (2018).
- [146] Salome Maro, Jan-Philipp Steghöfer, and Mirosław Staron. "Software traceability in the automotive domain: Challenges and solutions." In: *Journal of Systems and Software* 141 (2018).
- [147] Alistair Mavin and Neil Maiden. "Determining Socio-Technical Systems Requirements: Experiences with Generating and Walking Through Scenarios." In: *International Requirements Engineering Conference* (2003).
- [148] Alistair Mavin, Philip Wilkinson, Adrian Harwood, and Mark Novak. "EARS (Easy Approach to Requirements Syntax)." In: *International Requirements Engineering Conference* (2009).
- [149] Alistair Mavin, Philip Wilkinson, Sabine Teufl, Henning Femmer, Jonas Eckhardt, and Jakob Mund. "Does Goal-Oriented Requirements Engineering Achieve Its Goal?" In: *International Requirements Engineering Conference* (2017).
- [150] Alistair Mavin, Philip Wilksinson, Sarah Gregory, and Eero Uusitalo. "Listens learned (8 lessons learned applying EARS)." In: *International Requirements Engineering Conference* (2016).
- [151] Richard E. Mayer. "Multimedia learning." In: *Psychology of learning and motivation* 41 (2002).
- [152] Fergal Mc Caffery, John Burton, Valentine Casey, and Alec Dorling. "Software Process Improvement in the Medical Device Industry." In: *Encyclopedia of Software Engineering* 1 (2010).

- [153] Jan Mendling, Henrik Leopold, and Fabian Pittke. "25 Challenges of Semantic Process Modeling." In: *International Journal of Information Systems and Software Engineering for Big Companies* 1.1 (2014).
- [154] Tom Mens and Pieter Van Gorp. "A Taxonomy of Model Transformation." In: *Electronic Notes in Theoretical Computer Science* 152 (2006).
- [155] Farid Meziane, Nikos Athanasakis, and Sophia Ananiadou. "Generating Natural Language specifications from UML class diagrams." In: *Requirements Engineering* 13.1 (2008).
- [156] Farid Meziane and Sunil Vadera. "Obtaining E-R Diagrams Semi-Automatically from Natural Language Specification." In: *International Conference on Enterprise Information Systems* (2004).
- [157] Luisa Mich. "NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA." In: *Natural Language Engineering* 2.2 (1996).
- [158] Luisa Mich, Mariangela Franch, and Pierluigi Novi Inverardi. "Market research for requirements analysis using linguistic tools." In: *Requirements Engineering* 9.2 (2004).
- [159] Hyun-Seok Min. "Traceability Guideline for Software Requirements and UML Design." In: *International Journal of Software Engineering and Knowledge Engineering* 26.01 (2016).
- [160] Modelica Association. *Modelica*. (<https://www.modelica.org/>). Accessed: 2018-05-14.
- [161] Yaniv Mordecai and Dov Dori. "Model-Based Requirements Engineering: Architecting for System Requirements with Stakeholders in Mind." In: *International Systems Engineering Symposium* (2017).
- [162] Gunter Mussbacher et al. "The Relevance of Model-Driven Engineering Thirty Years from Now." In: *International Conference on Model Driven Engineering Languages and Systems* (2014).
- [163] Farhana Nazir, Wasi Haider Butt, Muhammad Waseem Anwar, and Muazzam A. Khan Khattak. "The applications of natural language processing (NLP) for software requirement engineering-a systematic literature review." In: *International Conference on Information Science and Applications* (2017).
- [164] Colin J. Neill and Phillip A. Laplante. "Requirements Engineering: The State of the Practice." In: *IEEE Software* 20.6 (2003).
- [165] Joaquín Nicolás and Ambrosio Toval. "On the generation of requirements specifications from software engineering models: A systematic literature review." In: *Information and Software Technology* 51.9 (2009).

- [166] Nan Niu and Steve Easterbrook. "Extracting and Modeling Product Line Functional Requirements." In: *International Requirements Engineering Conference* (2008).
- [167] Ariadi Nugroho and Michel R.V. Chaudron. "A Survey into the Rigor of UML Use and its Perceived Impact on Quality and Productivity." In: *International Symposium on Empirical Software Engineering and Measurement* (2008).
- [168] Bashar Nuseibeh and Steve Easterbrook. "Requirements engineering: a roadmap." In: *Conference on the Future of Software Engineering* (2000).
- [169] Object Management Group (OMG). *Foundational UML Subset*. (<https://www.omg.org/spec/FUML/About-FUML/>). Accessed: 2018-05-15.
- [170] Object Management Group (OMG). *MOF Model to Text Transformation Language (MOFM2T), Version 1.0*. OMG Document Number formal / 2008-01-16 (<https://issues.omg.org/issues/spec/MOFM2T/1.0/>). 2008.
- [171] Object Management Group (OMG). *Business Process Model And Notation, Version 2.0*. OMG Document Number formal / 2011-01-03 (<https://www.omg.org/spec/BPMN/2.0/About-BPMN/>). 2011.
- [172] Object Management Group (OMG). *OMG Unified Modeling Language (OMG UML), Version 2.5*. OMG Document Number formal / 2015-03-01 (<https://www.omg.org/spec/UML/2.5/>). 2015.
- [173] Object Management Group (OMG). *Requirements Interchange Format (ReqIF), Version 1.2*. OMG Document Number formal / 2016-07-01 (<https://www.omg.org/spec/ReqIF/1.2/>). 2016.
- [174] Object Management Group (OMG). *Action Language for Foundational UML (ALF), Version 1.1*. OMG Document Number formal / 2017-07-04 (<https://www.omg.org/spec/ALF/1.1/>). 2017.
- [175] Object Management Group (OMG). *OMG Systems Modeling Language (OMG SysML), Version 1.5*. OMG Document Number formal / 2017-05-01 (<https://www.omg.org/spec/SysML/1.5/>). 2017.
- [176] Open Services for Lifecycle Collaboration. (<https://open-services.net/>). Accessed: 2018-05-29.
- [177] W. J. Orlikowski. "Using technology and constituting structures: A practice lens for studying technology in organizations." In: *Organisation Science* 11.4 (2000).

- [178] Cristina-Claudia Osman and Paula-Georgiana Zalhan. "From Natural Language Text to Visual Models: A survey of Issues and Approaches." In: *Informatica Economica* 20.4 (2016).
- [179] Scott P. Overmyer, Benoit Lavoie, and Owen Rambow. "Conceptual Modeling through Linguistic Analysis Using LIDA." In: *International conference on Software engineering* (2001).
- [180] James L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981.
- [181] Marian Petre. "UML in Practice." In: *International Conference on Software Engineering* (2013).
- [182] Marian Petre. "'No shit' or 'Oh, shit!': responses to observations on the use of UML in professional practice." In: *Software & Systems Modeling* 13.4 (2014).
- [183] Fredrik Pettersson, Martin Ivarsson, and Peter Öhman. "Automotive use case standard for embedded systems." In: *ACM SIGSOFT Software Engineering Notes* 30.4 (2005).
- [184] Fabian Pittke. *Linguistic Refactoring of Business Process Models*. Dissertation. Logos Verlag Berlin GmbH, 2016.
- [185] Klaus Pohl. *Requirements engineering: fundamentals, principles, and techniques*. Springer, 2010.
- [186] Alexander Pretschner, Manfred Broy, Ingolf H. Kruger, and Thomas Stauner. "Software Engineering for Automotive Systems: A Roadmap." In: *Future of Software Engineering* (2007).
- [187] Nils Przigoda, Christoph Hilken, Robert Wille, Jan Peleska, and Rolf Drechsler. "Checking Concurrent Behavior in UML / OCL Models." In: *International Conference on Model Driven Engineering Languages and Systems* (2015).
- [188] Virgilio Quintana, Louis Rivest, Robert Pellerin, Frédérick Venne, and Fawzi Kheddouci. "Will Model-based Definition replace engineering drawings throughout the product lifecycle? A global perspective from aerospace industry." In: *Computers in Industry* 61.5 (2010).
- [189] Rick Rabiser, Wolfgang Heider, Christoph Elsner, Martin Lehofer, Paul Grünbacher, and Christa Schwanninger. "A Flexible Approach for Generating Product-Specific Documents in Product Lines." In: *International Conference on Software Product Lines* (2010).
- [190] Gary L. Ragatz, Robert B. Handfield, and Thomas V. Scannell. "Success Factors for Integrating Suppliers into New Product Development." In: *Journal of Product Innovation Management* 14.3 (1997).

- [191] Erhard Rahm and Philip A. Bernstein. "A survey of approaches to automatic schema matching." In: *The VLDB Journal* 10.4 (2001).
- [192] Amit Raj, T.V. Prabhakar, and Stan Hendryx. "Transformation of SBVR Business Design to UML Models." In: *India Software Engineering Conference* (2008).
- [193] Bala Ramesh, Curtis Stubbs, and Michael Edwards. "Lessons learned from implementing requirements traceability." In: *Journal of Defense Software Engineering* 8.4 (1995).
- [194] Balasubramaniam Ramesh and Matthias Jarke. "Toward Reference Models for Requirements Traceability." In: *IEEE transactions on software engineering* 27.1 (2001).
- [195] Anand Ranganathan and Roy H. Campbell. "What is the Complexity of a Distributed Computing System?" In: *Complexity* 12.6 (2007).
- [196] Gianna Reggio, Maurizio Leotta, and Filippo Ricca. "Who Knows / Uses What of the UML: A Personal Opinion Survey." In: *International Conference on Model Driven Engineering Languages and Systems* (2014).
- [197] Gianna Reggio, Maurizio Leotta, Filippo Ricca, and Diego Clerissi. "What are the used Activity Diagram Constructs? A survey." In: *International Conference on Model-Driven Engineering and Software Development* (2014).
- [198] Hamzah Ritchi, Mieke Jans, Jan Mendling, and Hajo Reijers. "The influence of business process representation on performance of different task types." In: *Journal of Information Systems* 9.3 (201x).
- [199] Marcel Robeer, Garm Lucassen, Jan Martijn E.M. van der Werf, Fabiano Dalpiaz, and Sjaak Brinkkemper. "Automated Extraction of Conceptual Models from User Stories via NLP." In: *International Requirements Engineering Conference* (2016).
- [200] Christopher L. Robinson-Mallett. "An Approach on Integrating Models and Textual Specifications." In: *International Model-Driven Requirements Engineering Workshop* (2012).
- [201] Raphael De A. Rodrigues, Márcio De O. Barros, Kate Revoredo, Leonardo G. Azevedo, and Henrik Leopold. "An Experiment on Process Model Understandability Using Textual Work Instructions and BPMN Models." In: *Brazilian Symposium on Software Engineering* (2015).
- [202] Nick Russell, Wil M.P. van der Aalst, Arthur H.M. Ter Hofstede, and Petia Wohed. "On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling." In: *Asia-Pacific Conference on Conceptual Modelling* (2006).

- [203] Kevin Ryan. "The Role of Natural Language in Requirements Engineering." In: *International Symposium on Requirements Engineering* (1993).
- [204] Christian Salzmänn and Thomas Stauner. "Automotive Software Engineering An emerging application domain for software engineering." In: *Languages for System Specification: Selected Contributions on UML* (2004).
- [205] Josep Sànchez-Ferreres, Josep Carmona, and Lluís Padró. "Aligning Textual and Graphical Descriptions of Processes Through ILP Techniques." In: *International Conference on Advanced Information Systems Engineering* (2017).
- [206] João Santos, Ana Moreira, Vasco Amaral, and Uirá Kulesza. "Generating Requirements Analysis Models from Textual Requirements." In: *International Workshop on Managing Requirements Knowledge* (2008).
- [207] Patrizia Scandurra, Andrea Arnoldi, Tao Yue, and Marco Dolci. "Functional Requirements Validation by transforming Use Case Models into Abstract State Machines." In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing* (2012).
- [208] Tim Schattkowsky and Alexander Förster. "On the Pitfalls of UML 2 Activity Modeling." In: *International Workshop on Modeling in Software Engineering* (2007).
- [209] Tim Schattkowsky, Wolfgang Müller, and Achim Rettberg. "A Generic Model Execution Platform for the Design of Hardware and Software." In: *UML for SoC design* (2005).
- [210] Robert Scheffler, Sergej Koch, Gregor Wrobel, Matthias Pleßow, Christian Buse, and Bernd-Arno Behrens. "Modelling CAD Models: Method for the Model Driven Design of CAD Models for Deep Drawing Tools." In: *International Conference on Model-Driven Engineering and Software Development* (2016).
- [211] Bran Selic. "The Pragmatics of Model-Driven Development." In: *IEEE Software* 20.5 (2003).
- [212] Matt Selway, Georg Grossmann, Wolfgang Mayer, and Markus Stumptner. "Formalising natural language specifications using a cognitive linguistic/configuration based approach." In: *Information Systems* 54 (2015).
- [213] Shane Sendall and Jochen Küster. "Taming Model Round-Trip Engineering." In: *Workshop on Best Practices for Model-Driven Software Development* (2004).
- [214] Atif Shah, Mohamed Ali Alasow, Faisal Sajjad, and Jawad Javed Akbar Baig. "An Evaluation of Software Requirements Tools." In: *International Conference on Intelligent Computing and Information Systems* (2017).

- [215] Ernst Sikora, Bastian Tenbergen, and Klaus Pohl. "Industry needs and research directions in requirements engineering for embedded systems." In: *Requirements Engineering* 17.1 (2012).
- [216] Michał Śmiałek, Jacek Bojarski, Wiktor Nowakowski, Albert Ambroziewicz, and Tomasz Straszak. "Complementary Use Case Scenario Representations Based on Domain Vocabularies." In: *International Conference on Model Driven Engineering Languages and Systems* (2007).
- [217] Neal Snooke and Chris Price. "Model-driven Automated Software FMEA." In: *Reliability and Maintainability Symposium* (2011).
- [218] Stéphane S. Somé. "Supporting use case based requirements engineering." In: *Information and Software Technology* 48.1 (2006).
- [219] Tony Spiteri Staines. "Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets." In: *International Conference and Workshop on the Engineering of Computer Based Systems* (2008).
- [220] International Organization for Standardization. *ISO/DIS 26262 - Road vehicles — Functional safety*. 2009.
- [221] Harald Störrle. "Semantics and Verification of Data Flow in UML 2.0 Activities." In: *Electronic Notes in Theoretical Computer Science* 127.4 (2005).
- [222] Harald Störrle. "How are Conceptual Models used in Industrial Software Development?: A Descriptive Survey." In: *International Conference on Evaluation and Assessment in Software Engineering* (2017).
- [223] Harald Störrle and Jan Hendrik Hausmann. "Towards a Formal Semantics of UML 2.0 Activities." In: *Software Engineering* (2005).
- [224] Kalaivani Subramaniam, Dong Liu, Behrouz Homayoun Far, and Armin Eberlein. "UCDA Use Case Driven Development Assistant Tool for Class Model Generation." In: *Conference on Software Engineering and Knowledge Engineering* (2004).
- [225] The International Council on Systems Engineering (INCOSE). "INCOSE SE Vision 2020." In: *INCOSE 2007 Symposium* (2007).
- [226] The Institute of Electrical and Electronics Engineers, Inc. *ISO / IEC / IEEE 29148:2011, Systems and Software Engineering – Life cycle processes – Requirements Engineering*. 2011.
- [227] The MathWorks, Inc. *Matlab/Simulink*. (<https://www.mathworks.com/products/simulink.html>). Accessed: 2018-05-14.
- [228] Saurabh Tiwari, Santosh Singh Rathore, Abhijeet Singh, Abhinav Singh, and Atul Gupta. "An Approach to Generate Actor-Oriented Activity Charts from Use Case Requirements." In: *Asia-Pacific Software Engineering Conference* (2012).



- [229] Suichi Tsujimoto and Haruo Asada. "Understanding Multi-articled Documents." In: *International Conference on Pattern Recognition* (1990).
- [230] Oktay Türetken, Onur Su, and Onur Demirörs. "Automating Software Requirements Generation from Business Process Models." In: *Conference on the Principles of Software Engineering* (2004).
- [231] Muhammad Usman and Aamer Nadeem. "Automatic Generation of Java Code from UML Diagrams using UJECTOR." In: *International Journal of Software Engineering and Its Applications* 3.2 (2009).
- [232] VDA QMC Working Group 13 / Automotive SIG. *Automotive SPICE Process Assessment / Reference Model*. 2015.
- [233] Verein Deutscher Ingenieure. *VDI/VDE 3694 Lastenheft/Pflichtenheft für den Einsatz von Automatisierungssystemen*. 2014.
- [234] Andreas Vogelsang, Sebastian Eder, Georg Hackenberg, Maximilian Junker, and Sabine Teufl. "Supporting concurrent development of requirements and architecture: A model-based approach." In: *International Conference on Model-Driven Engineering and Software Development* (2014).
- [235] Matthias Weber and Joachim Weisbrod. "Requirements Engineering in Automotive Development - Experiences and Challenges." In: *Joint International Conference on Requirements Engineering* (2002).
- [236] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Haldal. "Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem?" In: *International Conference on Model Driven Engineering Languages and Systems* (2013).
- [237] Stefan Wiesner, Margherita Peruzzini, Jannicke Baalsrud Hauge, and Klaus-Dieter Thoben. "Requirements engineering." In: *Concurrent Engineering in the 21st Century* (2015).
- [238] Stefan Winkler and Jens von Pilgrim. "A survey of traceability in requirements engineering and model-driven development." In: *Software & Systems Modeling* 9.4 (2010).
- [239] Rebekka Wohlrab, Patrizio Pelliccione, Eric Knauss, and Sarah C. Gregory. "The Problem of Consolidating RE Practices at Scale: An Ethnographic Study." In: *International Working Conference on Requirements Engineering: Foundation for Software Quality* (2018).
- [240] Adam Wyner, Krasimir Angelov, Guntis Barzdins, Danica Damjanovic, Brian Davis, Norbert Fuchs, Stefan Hoefler, Ken Jones, Kaarel Kaljurand, Tobias Kuhn, et al. "On Controlled Natural Languages: Properties and Prospects." In: *International Workshop on Controlled Natural Language* (2009).

- [241] Eric Yu, Eric Dubois, and John Mylopoulos. "From Organization Models to System Requirements A Cooperating Agents Approach." In: *Conference on Cooperative Information Systems* (1995).
- [242] Tao Yue, Shaukat Ali, and Lionel Briand. "Automated Transition from Use Cases to UML State Machines to Support State-Based Testing." In: *European Conference on Modelling Foundations and Applications* (2011).
- [243] Tao Yue, Lionel C. Briand, and Yvan Labiche. "A systematic review of transformation approaches between user requirements and analysis models." In: *Requirements Engineering* 16.2 (2011).
- [244] Tao Yue, Lionel C. Briand, and Yvan Labiche. "Facilitating the transition from use case models to analysis models: Approach and experiments." In: *ACM Transactions on Software Engineering and Methodology* 22.1 (2013).
- [245] Iyad Zikra, Janis Stirna, and Jelena Zdravkovic. "Analyzing the Integration between Requirements and Models in Model Driven Development." In: *Enterprise, Business-Process and Information Systems Modeling* (2011).