Thesis for The Degree of Licentiate of Engineering

Addressing Traceability Challenges in the Development of Embedded Systems

Salome Maro

Division of Software Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and Göteborg University
Göteborg, Sweden, 2017

Addressing Traceability Challenges in the Development of Embedded Systems

Salome Maro

Technical Report No 164L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Software Engineering
Chalmers University of Technology and Göteborg University
Göteborg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Göteborg, Sweden 2017.

To my family

Abstract

Context: Currently, development efforts in embedded systems development lead to a large number of interconnected artifacts. Traceability enables understanding and managing these artifacts as they evolve. However, establishing traceability is not a trivial task, it requires the development organization to plan how traceability will fit into its processes and provide tools to support traceability establishment. In practice, guidelines for how traceability should be established are lacking. Therefore, companies struggle with establishing traceability and making the most of traceability once it is established.

Objective: The overall objective of this research is to improve traceability processes and tools for embedded systems development. In this thesis, we started with first understanding the domain and practical traceability challenges and also investigated how traceability tools can be improved.

Method: Since establishing traceability is a practical problem, our research is conducted in close collaboration with industry partners. We conducted qualitative empirical studies to understand which traceability challenges exist in reality and designed solutions for some of these challenges. Concretely, we used action research, case study and design science methods for the different studies.

Results: Our studies show that establishing traceability in practice still has several challenges, the most prominent ones being: the manual work of establishing traceability is high; the engineers responsible for creating the links perceive it as an overhead; lack of tools to enable using traceability; lack of methods and tools to measure its quality; no universal standards for traceability to be shared and exchanged and it is difficult to measure the return on investment of establishing traceability.

To reduce the amount of manual work needed to maintain traceability links, we designed guidelines that can be followed by tool developers. We also show the feasibility of a configurable and extendable traceability management tool through a prototype implementation.

Contributions: As part of this thesis, we have elicited persistent traceability challenges in development of embedded systems development. This list of challenges can also be used by other researchers who are interested in the topic of traceability for embedded systems development. As a first initiative towards solving these challenges, we propose important factors and guidelines for traceability tool developers and organizations that need to acquire traceability tools. Lastly, we have demonstrated the feasibility of these factors and guidelines through a prototype implementation. This implementation is open source and available for industry to use in their development and for other researchers to use for studies and extend the tool.

Keywords Traceability, Software Traceability, Embedded Systems

Acknowledgment

I would like to thank my supervisors, Jan-Philipp Steghöfer and Miroslaw Staron. To Jan-Philipp, thanks for the ideas, guidance, discussions and extensive feedback (including code reviews). To Miroslaw, thanks for providing great support, feedback and steering me to the right research directions.

I would also like to thank my previous supervisor Matthias Tichy for the support provided during the time he acted as my supervisor.

Lots of thanks go to Anthony Anjorin, for the development effort and great ideas on Capra (the tool). To Rebekka Wohlrab, thanks for being such a hard worker and a good colleague to work with.

To my office mate, Grischa Liebel, thanks for the good discussions and answering countless questions from me.

I would also like to thank my fellow PhD students, faculty and administration of the software engineering division for being great collaborators and providing help whenever needed. Special thanks go to Rodi and Truong, the good and helpful neighbors.

Behind every successful person, there is a supporting family and a group of great friends. I would like to thank my parents (Honest and Radegunda Maro) and my sisters (Judy, Rose and Vicky), for their continuous support and encouragement. To my friends, especially Jacky(Boko) and Asna, thanks for the love and encouragement. Most important, to Seif Hamad, my love, thanks for your support, encouragement and patience.

Majority of the work reported in this thesis was conducted as part of the AMALTHEA4Public project, funded by the ITEA EUREKA Cluster program.

List of Publications

Appended publications

This thesis is based on the following publications:

[A] S. Maro, J.-P. Steghöfer, A. Anjorin, M. Tichy, L. Gelin "On Integrating Graphical and Textual Editors for a UML Profile Based Domain Specific Language: An Industrial Experience"
13th International Conference on Software Language Engineering(SLE 2015), Pittsburg, USA, October 23-27, 2015.

[B] S. Maro, M. Staron, J.-P. Steghöfer "Persisting Software Traceability Challenges in the Automotive Domain"
In submission to Journal of Systems and Software.

[C] S. Maro, A. Anjorin, R. Wohlrab, J.-P. Steghöfer "Traceability Maintenance: Factors and Guidelines"
31st International Conference on Automated Software Engineering (ASE 2016), Singapore, Singapore, September 3-7, 2016.

[D] S. Maro, J.-P. Steghöfer "Capra: A Configurable and Extendable Traceability Management Tool"
24th International Conference on Requirements Engineering (RE2016), Beijing, China, September 12 - 16, 2016.

Other publications

The following publications were published during my PhD studies. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[a] R. Wohlrab, J.-P. Steghöfer, E. Knauss, S. Maro, A. Anjorin "Collaborative Traceability Management: Challenges and Opportunities"
24th International Conference on Requirements Engineering (RE 2016), Beijing, China, September 12 - 16, 2016.

[b] M. Trei, S. Maro, J.-P. Steghöfer, T. Peikenkamp "An ISO 26262 Compliant Design Flow and Tool for Automotive Multicore Systems"
17th International Conference on Product Focused Software Process Improvement (PROFES, 2016), Trondheim, Norway, November 22-24, 2016.

[c] S. Maro, M. Staron, J.-P. Steghöfer "Challenges of Establishing Traceability in the Automotive Domain"
9th International Conference on Software Quality (SWQD 2017), Vienna, Austria, January 17-20, 2017.

Research Contribution

The work reported in this thesis was done in collaboration with other people. For paper A and B, I took responsibilities for the study design, implementation and analysis of the results. I also wrote a majority of the text in the papers and the co-authors acted as internal reviewers through giving feedback.

In paper C, I was responsible for collecting data on the tools and interviewing tool experts. I also wrote a majority of the sections referring to the tools.

Paper D, reports on a tool that was developed with me in collaboration with other developers. I am currently the project leader for the tool development.

Contents

Chapter 1

Introduction

Embedded systems have been around for a long time, with the first mass release dated back to 1961: the Autonetics D-17 guidance computer developed at MIT [1]. But with the decrease in the price and size of hardware, and increased performance, embedded systems are now present in more and more domains. Compared to the Autonetics D-17 which only had one programmable computer, today, a modern car contains contains software distributed over 100 Electronic Control Units (ECUs).The adoption and growth of embedded systems is not only observed in the automotive domain but also in other domains such as health-care and telecommunications.

Embedded systems are becoming increasingly complex and used for safety-critical functions such as adaptive cruise control systems in the automotive domain. With this increase in complexity of the systems, during development, a large amount of artifacts such as requirements, models, code and tests are being produced. For instance, the software in a modern car consist of around 100 million lines of code, while the requirements were already up to 20,000 pages in 2004 [2]. This complexity can also be observed in other domains like the telecommunication domain where the software in the telephone switches contains around 200 million lines of code [3] while their requirements were already over 10,000 in 2008 [4].

Managing such large amounts of artifacts is difficult for developers and other stakeholders involved in the system development. This is because these artifacts do not exist in pure isolation but are related to each other and consistency needs to be ensured during their evolution. Traceability which is the ability to relate different development artifacts is therefore very important as it can be used to reason about the relationships between the different artifacts. Some of the benefits of traceability are: increasing program comprehension, facilitating impact analysis, facilitating tracking of project progress and supporting change propagation [5–8]. However for these benefits to be realized, a development organization needs to invest in establishing traceability. This means putting in place processes on how traceability links will be created, maintained and used and providing tools to support these activities. Establishing traceability in practice still remains a challenge as in many development organizations, traceability practices are poor, mainly due to lack of well-defined processes and tool support for establishing traceability links [9, 10]. The overall goal of this

research is therefore to *improve traceability processes and tools for embedded systems development*. As first steps towards achieving this overall goal, in this licentiate thesis we will address the following goals:

Goal 1: Investigate current traceability challenges in the development of embedded systems

Goal 2: Propose conceptual tool solutions that can solve the identified challenges

Goal 3: Develop prototypes that demonstrate how traceability solutions can be implemented

In order to improve the practices, there is a need to first understand the current state of practice within the embedded systems domain and understand the challenges that need to be dealt with to improve the current status. This is what Goal 1 will address. With these challenges as a starting point, in Goal 2, we investigate and propose solutions for some of these challenges. While proposing conceptual solutions is a valuable contribution, to be able to transfer these solutions to industry, there is a need to test that the solutions actually work in practice. This will be addressed in Goal 3, where we develop a traceability management tool based on requirements from industry practitioners developing embedded systems. The goal of improving traceability *processes* will be addressed in our upcoming research.

The rest of this Chapter is structured as follows: First a background on the topic of traceability is given by Section 1.1, 1.2 and 1.3 where we discuss definitions of traceability, traceability activities and tools for traceability. This is followed by a discussion on existing traceability challenges and motivation for our research discussed in Section 1.4. The scope and methodology for the research and how the threats to validity were mitigated are reported in Section 1.5, 1.6, and 1.7 respectively. The contribution of the thesis are given in Section 1.8. Section 1.9 gives a discussion on how the research questions were answered and Section 1.10 outlines our future work.

1.1 Traceability Definition

In literature, there are a number of definitions of traceability. This section gives an overview of the most cited traceability definitions, analyses the definitions and gives our definition of traceability that is used in this thesis. This analysis is summarized in Table 1.1.

Gotel et al. [11] give a general definition of traceability as "the potential for traces (a specified triplet of elements comprising: a source artifact, a target artifact and a trace link associating the two artifacts) to be established (created and maintained) and used". The authors define a trace artifact as a "traceable unit of data". This definition explicitly mentions the use of trace links implying that links should only be established if they will be used. The shortcoming of this definition is that it is recursive as traceability is defined using the term traces and trace artifact is defined using the word traceable.

In line with the Gotel et al. [11] definition, the Center of Excellence for Software and System Traceability (COEST) [12] defines *software* traceability as "the ability to interrelate any uniquely identifiable software engineering artifact to any other, maintain required links over time, and use the resulting

network to answer questions of both the software product and its development process". This definition states that the artifacts need to be unique which is a characteristic of a traceable artifact [13] and also that the traces should be maintained over time and used in the development process. However from the definition, it is not clear what the authors mean by "maintain required links". This may imply that some links that are not required may be created but not maintained. Also the "over time" is very generic and may be interpreted in several ways, e.g., to mean forever or for a certain period of time.

Another definition is by Spanoudakis and Zisman [6] who define traceability as "the ability to relate artifacts created during the development of a software system to describe the system from different perspectives and levels of abstraction with each other, the stakeholders that have contributed to the creation of the artifacts, and the rationale that explains the form of the artifacts". Like the previous definition, this one also stresses the use of traceability links and even which these uses are; for instance being able to explain the rationale of the artifacts. However, making the uses explicit in the definition gives an impression that these are the only uses of traceability while in reality there are many more, for instance facilitating tracking of project progress and reuse of artifacts [5].

Older definitions are given in the IEEE standard glossary of software engineering terminology [14] which gives two definitions of traceability:

[a] "The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match". This definition does not mention anything suggesting that these relationships must be useful. Additionally, even though the example given may give an idea of what "product" means in this case, the word "product" may be interpreted as a complete software or system implying a certain level of granularity for the traced artifacts.

[b] "The degree to which each element in a software development product establishes its reason for existing; for example, the degree to which each element in a bubble chart references the requirement that it satisfies". Like the first definition, this one also does not mention that the relationships should be useful. On the other hand, this definition uses the term "element" which could refer to a more fine grained granularity compared to the term "product" in the previous definition.

There are also definitions which are explicitly from a requirements perspective. Gotel & Finkelstein [15] define *requirements traceability* as "the ability to follow the life of a requirement in both forwards and backwards direction (i.e., from its origins through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)." This is the only definition that discusses refinement and iterations of artifacts. This implies not only being able to "follow" a requirement to other artifacts but also being able to follow and track the different versions. However this definition is only focused on requirements. It does not explicitly mention the need to connect intermediate artifacts that

Table 1.1: Analysis of Traceability Definitions

Source	Trace What?	To what?	Trace when?	Why?
[11]	Development artifact	Development artifact	–	To be used
[12]	uniquely identifiable software engineering artifact	uniquely identifiable software engineering artifact	over time	use the resulting network to answer questions of both the software product and its development process
[6]	Development artifact	Development artifact		to describe the system, the stakeholders and the rationale of artifacts
[14]	product of software development process	product of software development process	–	–
[14]	element in a software development process	element in a software development process	–	–
[15]	Requirements	development artifacts	Development life cycle including use in the field	–
[16]	Requirements	development artifacts	Development life cycle	–

may not be related to requirements. Also the definition only informs about being able to follow the life of a requirement but does not mention anything about the use of traceability. A similar definition that is requirements-oriented is that by Spanoudakis [16] which defines requirements traceability as the "ability to relate requirements specifications with other artifacts created in the development life cycle of a software system"

Analyzing all these definitions, we can deduce that there are three aspects that are important to be able to define traceability. These are: *artifacts involved, which point in the development and the purpose of the links.* We can reduce this into *what*, *when* and *why* questions (Table 1.1). The question of *how* traceability links are established is orthogonal to all of these and thus not included in this classification.

Based on this analysis, we deduce a definition that will be used throughout the thesis. Since the scope of our research is not only software development but system development, we need to also consider other artifacts created during system development that are not software related, for example hardware models. Therefore we define *system* traceability as follows:

System Traceability. *The ability to relate uniquely identifiable system engineering artifacts created and evolved during the development of a system, maintain these relationships throughout the development life cycle and use them to facilitate system development activities.*

System development artifacts in this case include requirements, design

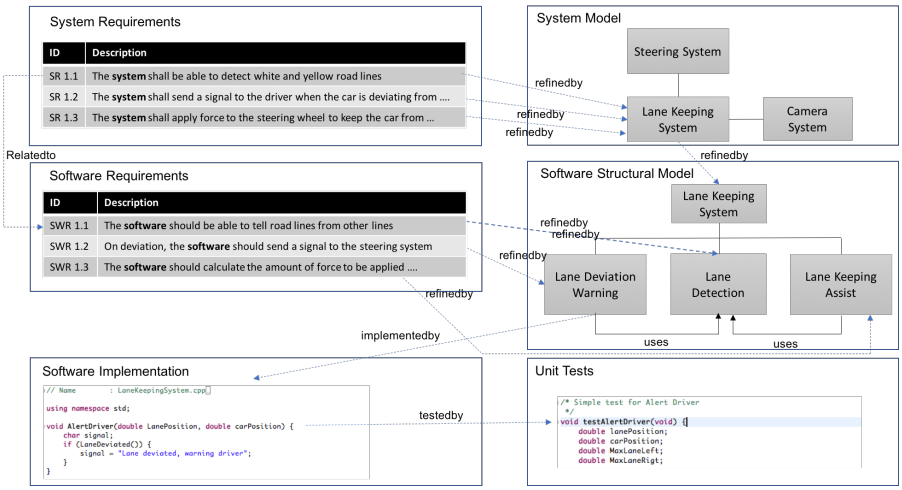

Figure 1.1: A simplified example of artifacts in a lane keeping system and traceability links involved.

models, behavior models, hardware models, code, test cases, test results, stakeholders and all other artifacts that are related to the system. We stress inclusiveness of development artifacts because in many cases there have been confusion on the scope of traceability where most practitioners seem to think that it is only limited to requirements artifacts [17]. We also stress the usefulness of the links because it is possible to create links to arbitrary artifacts but this is not only a waste of time when the links have no purpose, but also creates a lot of noise and may make the actual useful links difficult to find and use. This definition is similar to the one in [12], but substitutes software with system to include other artifacts such as hardware models. Substitution of software with system has also been suggested by the authors in [12] for practitioners or researchers interested in system traceability rather than software traceability. Our definition also substitutes "over time" with development life cycle to make the *when* explicit and includes the word "evolved" to mean that artifacts can have different versions and these versions should be traceable.

1.1.1 Example

Let us take an example of a lane keeping system from the automotive domain. This is a system that is used to detect lanes on the road and help the driver keep the car within the lane by sending warnings when deviation occurs and applying a small force to steer the car back to the lane. During the development of such a system, the following artifacts may be produced:

- A high level description of the lane keeping system as a whole and which other systems it interacts with, e.g., the steering wheel. The results of this can be stored as textual requirements (Figure 1.1 top left) as well as a system model (Figure 1.1 top right). A system model is an abstract description of system components and how they are connected to each

other.

- The system model is then broken down into discipline-specific subsystems that can be handled by the different domains, for instance mechanical, electrical, electronic and software.

From a software perspective further artifacts produced are:

- Software requirements, which can be in form of free text (Figure 1.1 mid left) or formal models such as use case models.
- Design models such structural models (models that describe the software components and their connections) (Figure 1.1 mid right) and behavior models (models that describe the control flow of the software) such as state charts.
- Implementation in form of code (Figure 1.1 bottom left).
- Tests such as unit tests (Figure 1.1 bottom right) and integration tests.

Traceability in such a scenario will be the ability to relate a system requirement to its component realization in the system model, the software requirements it affects, its corresponding software components in the software model, the code that realizes this requirement and the tests validating the requirement. It may also include the ability to relate artifacts like change requests, task tickets and bug reports to the specific development artifact that concerns them, depending on if this information is later useful.

Figure 1.1 shows a simplified example of the development artifacts produced and the lines between the artifacts represent the traceability links that can be established between them. Note the arrows that are from the same element type to the same element type. These indicate that traceability links can also be between elements of the same type, for instance a relationship between a requirement and a requirement or a model element and a model element.

1.2 Traceability Activities

Several activities are involved in establishing traceability in practice. The development organization needs to plan which links will be created, how they will be created, how they will be maintained, how they will be used and how they will be checked for quality. All these activities also need to fit into the existing development process of the company, be it agile or plan-driven. For these activities to be efficiently conducted, it is not only important for the company to invest in defining the traceability process, but also make sure that tools to support these activities are available. This section gives an overview of traceability activities and how they depend on each other. A discussion of the traceability tools is given in Section 1.3.

To facilitate easy understanding of the activities, we derive a traceability model depicted in Figure 1.2. This model is inspired by a generic traceability process model by Gotel et al. [11] which has further been used in several traceability research projects to describe traceability activities. Our model contains the four activities from the model in [11] which are preparation &

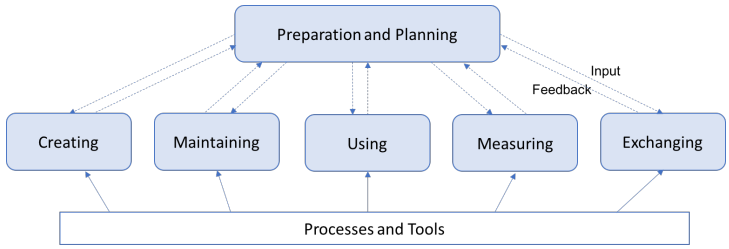

Figure 1.2: A traceability model showing the different categories of traceablity activities.

planning, creating, maintaining, and using traceability and two additional activities discovered during our research which are measuring and exchanging of traceability. This model has also been used as a basis for the study reported in Chapter 3.

The first activity in the model is preparation and planning for traceability. This activity includes all tasks involved in preparing and planning for a traceability strategy. These tasks include, but are not limited to, eliciting the needs of the development organization, searching for fitting solutions, acquiring tools, documenting and disseminating the traceability strategy. For a development organization to be able to succeed with traceability a proper strategy needs to be derived and followed. Traceability does not just happen, it has to be planned for and resources need to be allocated for the traceability activities. Planning a traceability strategy involves identifying what are the needs in the development organization and allocating resources including tools and man-hours. It is also important that the traceability strategy is assessed periodically and improved according to the needs of the organization [11]. The main input for the preparation and planning activity is the purpose for which traceability is established. This is inline with our definition that there should always be an intended use for the traceability links created. Reported examples of traceability purposes are: for certification purposes (e.g., in safety-critical industries) [18], for impact analysis [19], for change management [5], etc.

The second activity in the model is the creation of traceability links. Traceability links can be implicit or explicit. Implicit traceability refers to traceability that is established based on conventions. For instance a traceability link exists if a requirement ID is similar to an ID of a model element. While implicit traceability may seem easy to establish, it is hard to enforce conventions, as conventions can be violated leading to either no links at all or existence of a partial set of potentially erroneous traceability links [20]. Explicit traceability means that the links between the different artifacts are created as separate artifacts. For instance a traceability link may be represented as an element in a model containing information of artifacts that it connects. Explicit links are advantageous because compared to implicit links they are can be easily checked for quality with tools. Explicit links can also be provided as input to visualization tools for analysis purposes. Where implicit links exist, automation techniques such as information retrieval techniques can be used to make the links explicit. The quality of the resulting explicit links will depend on the

performance of the algorithms used and also on the quality of the implicit links (i.e., the conventions of creating implicit links and if they were followed) [21].

The third activity is maintaining the traceability links which means making sure that the links are up-to-date. As artifacts connected by traceability links evolve, the traceability links also need to be updated otherwise they quickly become outdated and therefore useless [22]. Maintenance of traceability links without tool support is difficult because artifacts constantly evolve. While Version Control Systems (VCS) can check for changes in the artifacts, traceability tools need to use these changes to also check if they affect the existing traceability links. Current traceability tools for instance IBM DOORS[1] provide a notification framework that allow users to be notified of changes to artifacts that are connected by traceability links. This makes it easier for the users to know which links they have to update.

The fourth activity in the model is using traceability. As mentioned in our definition of traceability, the links created need to be useful during the development of the system. Just having the traceability links in place is not enough. For the links to be useful, tool support is needed for navigation and visualization purposes. For instance if a product manager wants to see how many requirements already have tests, it will be very difficult to navigate through all requirements one by one to their tests. The traceability tool should be able to provide this report based on a few clicks. The traceability strategy therefore should also elicit how these links will be used and provide the required tool support.

The fifth activity in the model is measuring the quality of traceability links. Since traceability links are used to facilitate other development activities like change impact analysis, it is important that the maintained set of links is both correct and complete. To ensure this, there needs to be methods in place that can enable the development organization to measure the quality of their links. There are two relevant metrics for traceability, completeness and correctness. The definition of completeness varies and is defined differently in different development organizations. For instance a completeness definition could be that every test has a link to a requirement. This can then be measured by checking if indeed all tests are linked to requirements. Correctness on the hand is harder to check for, as it relies on domain-specific semantics and also human interpretation of the artifacts. Semantics can be formalized and checked by tools through definition of domain-specific traceability metamodels. For instance in the case of linking tests to requirements, a tool can check if the link actually connects a test and a requirement. What is difficult to check is if the requirement is actually related to the test it is connected to.

The sixth activity is exchanging traceability information. This refers to situations where different organizations or different departments in the organization need to exchange traceability information. Due to the numerous tools used in different departments and different companies, there is a need for the development organizations to plan for how traceability information will be shared and exchanged. This may mean agreeing to use certain tools that are compatible with each other. Other issues that need to be considered are technical and legal issues on whether the different organizations have access to artifacts that are traced to.

[1] `http://www-03.ibm.com/software/products/en/ratidoor`

1.3 Traceability Tools

As discussed in Section 1.2, tooling is an important aspect as it enables the different traceability activities to be carried out. There are several tools that are used for supporting traceability activities and according to [17] they can be classified into three main categories: dedicated traceability management tools, life cycle tools and general purpose tools. Based on our own research, we add a fourth category which is standalone traceability tools. This fourth category is also supported by [23] and [9]. The four categories are as follows:

Dedicated Requirements Management Tools – These are tools whose main purpose is requirement management. Most of these tools provide traceability between requirements. Even though the main functionality of the tools are requirements management, they may have functionality to facilitate traceability from requirements to other artifacts such as design models, task tickets and code. An example of such a tool is IBM Rational DOORS[2], which is a requirements management tool with capabilities to connect to other artifacts providers through OSLC [24].

Life Cycle Tools – This category of tools is also known as Application Life Cycle Management (ALM) tools. They provide functionality for creation and management of the different development artifacts in the development life cycle. Life cycle tools support requirements management, design, implementation, testing and other development activities. All artifacts are stored in one repository and traceability between the different artifacts is provided. The advantages of life cycle tools is that it is comparably easier to establish traceability between different artifacts. However in many cases companies have a variety of tools in place due to fear of being dependent on one vendor, unwillingness to use a certain development methodology (e.g., model driven development) or the fact that developers prefer tools that are task focused over generic ones [17, 25, 26]. Examples of life cycle tools are Systemweaver[3] and IBM ALM[4].

General Purpose Tools – These are tools that are designed to be used for many purposes. These tools can also be used for traceability. For instance one of the most common general purpose tools used for traceability is Microsoft Excel. It is used to record the links between the different artifacts. The advantages of general purpose tools is that they are widely available and provide a cheaper traceability option for organizations. However, the disadvantage with such tools is that the tools do not scale for traceability maintenance. The tools are not aware of the actual artifacts and traceability is managed separately from the artifacts. The chances that the links and the actual artifacts become inconsistent is high.

Standalone Traceability Management Tools – These are tools that are built for the purpose of managing traceability only. Such tools need to be able to integrate and access artifacts from the development tool

[2] `http://www-03.ibm.com/software/products/en/ratidoor`
[3] `http://www.systemweaver.se`
[4] `https://www-01.ibm.com/software/rational/alm/`

chain to allow creation of traceability links. They also need to provide notification mechanisms when artifacts in the different tools change in order to facilitate maintenance of the links. These tools use tool integration technologies such as Open Services for Life Cycle Collaborations (OSLC) [24] or Eclipse Modeling Framework (EMF) [27] to integrate the different tools. The advantage of such tools is that they are configurable to include various artifacts and therefore do not force a company to change their current tools. The disadvantage is that the customization requires effort and may be costly especially in a company where tools are added or changed frequently. Examples of such tools are Yakindu Traceability[5] and Capra[6].

1.4 Traceability Challenges and Research Motivation

In this section we give an overview of traceability challenges that currently exist and describe the motivation for this research.

In 1994, Gotel and Finkelstein [28] reported an empirical study with over 100 practitioners that identified challenges of traceability in industry. 18 years later, Gotel and a group of traceability researchers published an updated version of challenges known as the grand challenges of traceability with a vision on how these challenges can be tackled by researchers [29]. The existence of these challenges serve as a motivation for our research. Since our overall objective is to improve traceability processes and tools, we aim to understand how these challenges manifest in practice and find solutions to address these challenges. Therefore, in this section, we give a summary of these grand challenges of traceability. A more detailed description can be found in Gotel et al. [29]. There are eight challenges described as follows:

Purposed: This challenge entails that traceability should be created for a reason (meaning that traceability is linked to the needs of the various stakeholders inside and outside the company).

Cost-effective: Currently there is a lack of methods for measuring the return of investment on establishing traceability. This means that for development organizations, it is hard to know which traceability practices and tools are the most economical and in which situations they deliver the most.

Valued: In order for traceability to be established and used in a development organization, all the stakeholders involved in the planning, creation, maintenance and use of the links need to value traceability. If traceability is not valued, it is treated as optional and of low priority and therefore poorly established.

Portable: In embedded systems development, a system needs to be developed across several departments and sometimes across development organiza-

[5] `https://www.itemis.com/en/yakindu/traceability/`
[6] `https://projects.eclipse.org/projects/modeling.capra`

tions. Traceability needs to be shared and efficiently exchanged between departments and organizations.

Trusted: For traceability to be used to facilitate development activities such as impact analysis or change management, the users need to be able to trust these links. Therefore it is important for the process and tools used for establishing traceability to be trusted and guarantee that the links are correct and complete.

Configurable: Different development organizations and even different projects in the same organization may have varying needs for traceability. The traceability purpose that the development organization or project has, drives these traceability needs. There is therefore no silver bullet solution for traceability. Traceability tools need to be able to handle different needs by allowing the link types and artifacts types to be configured according to the needs of the different stakeholders.

Scalable: Traceability solutions both in terms of processes and tools need to be able to scale to larger projects. This is very important for the use of traceability links. As projects grow larger the network of traceability links also grows. Tools and processes are needed to allow the end users to create, maintain and use traceability in such scenarios.

Ubiquitous: This challenge is referred to as the grand traceability challenge. It suggests that developers and other stakeholders should not think about establishing traceability links because traceability should be automatically established as they work. Traceability should always be there. This is the dream for perfect and seamless traceability that is established in the background with automation tools.

From the challenges above, the overall aim of traceability research to make traceability ubiquitous. And for this to be possible further research on solving these challenges is needed. As our overall goal is to improve traceability practices and tools, we will be contributing to solving some of these challenges. Our research is mainly directed towards ensuring that traceability is *configurable*, *scalable*, *trusted* and *cost-effective*. As the challenges are not completely orthogonal but interdependent, other challenges may also be solved by our research as well. For example, making sure traceability is *trusted* also contributes to making sure that it is *valued*, as people will use the tracebility links if they can trust them.

1.5 Research Scope

As mentioned in the previous section, our research is motivated by the fact that there are a number of traceability challenges that still need researchers' attention. In this section, we describe the research scope of this thesis and the research questions that we answer.

Our research started with a tool integration problem in which a textual editor needed to be introduced and integrated with an existing graphical editor. In this, we investigated how to integrate the editors in such a way that models

in the two notations (graphical and textual) do not become inconsistent as the models evolve. This is in general a traceability challenge, as in many companies the tool environment is heterogeneous, with different tools and notations used for the different tasks in the development life cycle [25]. To keep track of the different artifacts and their evolution in the different tools, traceability is important and to achieve traceability, the different tools need to be integrated [26, 30]. Therefore the first research question answered by this thesis is:

RQ1 How can editors be integrated in a development environment in a way that is cost-efficient and avoids inconsistencies as models evolve?

Answering **RQ1** led to the discovery of several challenges that companies encounter when integrating editors. While some of the challenges were specific to the tools we integrated, others were more general traceability challenges. For instance challenges of maintaining consistency between models and challenges of being able to transfer links from one model notation to another. With respect to the grand challenge, these challenges fall in the category of making traceability *cost-effective* (automatic means of ensuring consistency) which also implies *ubiquitous* and *portable* (how to maintain links between different notations). This led us expand our research focus and derive a second more general research question which is as follows:

RQ2 What are the current traceability challenges for embedded systems development?

The study conducted to answer **RQ2**, led to formation of further research questions. One of the challenges identified was that traceability is expensive to create and without proper maintenance, the links quickly become outdated. We therefore decided to investigate how traceability maintenance can be supported by traceability tools in order to make activities of maintaining traceability links less manual and less error prone. Our aim was to answer the following research question:

RQ3 What are the primary factors that affect how and to what extent a traceability management solution can provide traceability maintenance?

The study not only revealed which factors are important, but based on empirical evidence we were able to derive guidelines that can be followed by traceability tool developers in order to develop tools that will allow for efficient maintenance of traceability links. The answers to RQ3 also provided insights on challenges of *cost-effective*, *configurable* and *trusted* traceability.

Since we are working in close contact with industry, we collected requirements for a traceability solution with the aim of developing a traceability tool. The requirements collected were very diverse implying that the solution should be flexible and extendable. Therefore the last research question investigated in this thesis is directed towards solving the *configurable* and *scalable* challenges and is as follows:

RQ4 How can a traceability tool be implemented in such a way that it is configurable and extendable?

In relation to the three goals previously stated, **RQ1** and **RQ2** are directed towards achieving Goal 1 (Investigate current traceability challenges in the development of embedded systems). **RQ3** is towards achieving Goal 2 (Propose conceptual tool solutions that can solve the identified challenges) and **RQ4** is towards achieving Goal 3 (Develop prototypes that demonstrate how traceability solutions can be implemented).

1.6 Research Methodology

To achieve reliable results in research, research methods need to be followed systematically and justified as to why the selected methods fit the purpose of the research. In software engineering empirical research methods are common due to the fact that the field is a multi-disciplinary one that spans both social and technological aspects [31] and does not only involve tools but also humans who are using these tools. For this research, the studies have been conducted using qualitative empirical research methods. The overall research methodology is depicted in Figure 1.3.

The research started with a tool integration problem, specifically how graphical and textual modeling editors can be integrated and used in the same environment. Since this is a practical problem that had to be investigated in the company, we chose action research as our research method. Action research is a research method where the researchers and practitioners together identify a problem and implement several actions in order to solve the problem [32]. Action research is an iterative process which involves defining the problem, implementing an action and then evaluating the effect of this action on the problem. This is repeated until a solution to the problem has been found. Action research is mainly used to solve practical problems in industry and requires the researchers and practitioners to collaborate on solving the problem. We worked together with a telecommunication company that had this problem and performed several iterations to find a fitting solution. The details on how the study was conducted is reported in Paper A (Chapter 2).

As previously stated, the results of the first study led to an expansion of our research focus to traceability in general. To be able to answer RQ2, we conducted an exploratory study to systematically identify traceability challenges that exist both in practice and in literature. To identify the challenges and solutions in literature, we conducted a systematic literature review. Systematic literature reviews give the researcher knowledge on the research that already exists in the topic of interest and also helps to identify gaps for which further research can be directed to. When current literature reviews already exist in the topic, they can be used as starting points for researchers. As part of this research we aimed to find out which traceability challenges have been reported in literature and if there are any solutions proposed or implemented for these challenges. The overall aim was to compare these challenges and solutions with the ones found in practice in order to figure out which challenges need further investigations and which ones have already been solved at least from the literature point of view. Due to the existence of recent literature reviews on traceability [33–35], we conducted a systematic tertiary literature review on existing literature reviews to get this information. The tertiary literature

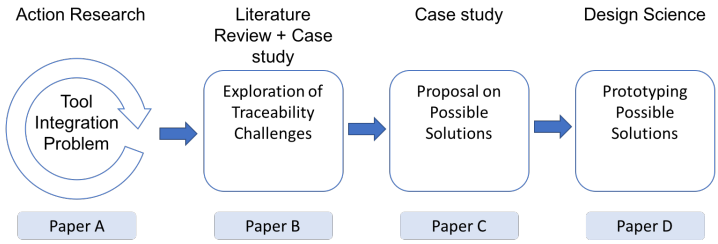

Figure 1.3: Research Methodology

review followed the same protocol as a systematic literature review [36].

Having the challenges from literature, we now wanted to investigate if these challenges are also found in practice. To achieve this, we conducted a case study with a large automotive supplier developing embedded systems. A case study is an empirical research method that is aimed at investigating contemporary phenomena in their context [37]. A case study can be descriptive, exploratory, explanatory or improving. We conducted an exploratory case study. An exploratory case study is aimed to collect information on what is happening so as to generate new research questions [37]. Our case study was conducted in a company that was following A-SPICE [38] and therefore was required to implement traceability. Selecting a company which is not required to have traceability may not have revealed the real challenges as the rigor in which traceability is established and used may have been low. The details of the tertiary literature review and the case study are reported in Paper B (Chapter 3).

From the identified challenges, we conducted further case studies in order to propose solutions that could work in practice. This led us to propose guidelines that can be followed by traceability tool developers in order to allow for more efficient traceability maintenance. These guidelines together with the details of how the study was conducted are reported in Paper C (Chapter 4).

To test the validity of our guidelines, we elicited further traceability requirements from industrial practitioners and used design science to develop a prototype tool for a traceability management solution based on the guidelines and requirements. Design science is defined as the design and investigation of artifacts in context [39]. Design science is an iterative process in which the researchers conduct several cycles of problem investigation, implementation of artifacts (e.g., software prototypes) that will solve the problem and evaluation of the artifacts in a given context to find out if the artifacts solve the problem. This prototype tool is now an open source Eclipse project [40] and the details of its implementation are reported in Paper D (Chapter 5).

1.7 Threats to Validity

In this section, we give an overview of the threats to validity for the studies included in this thesis. A more detailed description of the threats to validity can be found in the individual papers reporting the studies.

To answer RQ1, action research methodology was used (see Section 1.2). According to Checkland and Holwell [41], the main threat to validity with action research is the reliability of the study. Since action research can not achieve the same level of replicability as experiments, it is important for researchers to document their theories and methods and how the methods were used in the study. Even though the circumstances for the study can not be completely replicated, other researchers can get details on how the study was carried out and be able to repeat the study in other situations. In our study (reported in Paper A), we documented our research motivation, the tools and methods used and the environment in which the study was conducted.

Another threat to validity regarding RQ1 is generalizability. This refers to whether the results of the study can be generalized. Since we conducted the study in one company, we can not say that the results will hold for all other companies. To mitigate this threat we used standard technologies in our solution (e.g., EMF [27]) to ensure that our study can be generalized in situations that use similar standard technologies.

In order to answer RQ2, we conducted a tertiary literature review and a case study. Since a tertiary literature review is a systematic literature review conducted on other systematic literature reviews there is a chance that we missed some challenges that were not reported in the reviews. To mitigate this threat we used snowballing to include other papers that were reporting traceability challenges.

For the case study, there are four types of threats to validity as classified by Yin [42]: construct, internal, external and reliability.

Construct validity refers to whether the concepts of the study are understood by both the researcher and the participants in the same way. Since we used interviews and observations as our data collection techniques, we had to make sure that our concepts are understood by the participants. We achieved this by having meetings with the participants to discuss the study before the study began and made the intentions of the study clear through discussions in the meetings. For participants who were not available for the meeting, a similar discussion was held with them before their interviews. We also sent the interview guide a week before the study and performed member checking [43] with the results we obtained.

External validity questions to what extent the results of the study be generalized. To mitigate this we conducted the interviews with participants of different roles and departments to be able to triangulate our data sources. However, we cannot generalize the results before replicating the study since all the interviews were conducted in one company.

Reliability pertains to the replicability of the study. Replicating the setting in which the interviews and observations were conducted is not possible, but to ensure that the study can be repeated we documented the artifacts used in the study (interview guide, notes etc.) and also the process that was followed.

Internal validity questions if there other factors that have an influence on the studied factors that the researcher has no knowledge of. Since we base our conclusions on the interviews, there is a chance that the interviewees did not give honest answers with the fear that the answers would reflect badly on the company if the results were published. To mitigate this, we guaranteed the anonymity of both the interviewees and the company when publishing

the results. Complimenting the interviews with observations also helped to mitigate this threat.

The data to answer RQ3 was also collected through interviews, but in this case, multiple case studies were used. Construct validity and reliability was mitigated in the same manner as in the study used to answer RQ2. External validity was mitigated by having multiple cases from different domains. Since this study was exploratory, to mitigate internal validity, we showed our results to developers and expert users of traceability tools for confirmation. The participants were also guaranteed anonymity when we publish the results.

Design science was used to answer RQ4. We collected requirements for a traceability solution using the two focus groups. To mitigate construct validity and ensure reliability of the study, we used similar methods as those described for RQ2. To mitigate internal validity, we underwent several iterations of design, implementation and validation with our stakeholders. To mitigate external validity our focus groups involved participants from both industry and academia. However, all of our industrial participants were from the automotive domain. To make sure that our results are not specific to the automotive domain, we have open sourced our prototype to get feedback from other domains as well.

1.8 Contributions

In this section we outline the contribution of this thesis by stating which challenges our research is addressing. To recap, according to Gotel et al. [29] the grand challenges of traceability that require research effort are towards achieving purposed, cost-effective, configurable, trusted, scalable, portable, valued and ubiquitous traceability. Table 1.2 gives an overview of which challenges each paper touches upon. The contributions of each paper are discussed in the following sub-sections.

1.8.1 Paper A: On Integrating Graphical and Textual Editors for UML Profile Based Domain Specific Language: An Industrial Experience

In this paper, we investigated how to integrate graphical and textual editors so that they can be used by developers in the same environment. While there are several advantages of graphical modeling such as increasing the ease of understanding and reducing chances of errors [44], textual modeling also has advantages such as increasing the speed of editing and wide availability of editors [44]. To harness both advantages, both notations are needed and developers should be able to switch between the two notations. Unfortunately for UML modeling, most tools only provide graphical editors and we therefore investigated how to add a textual editor in such an environment and report on how this was done and the challenges encountered during the study.

The textual editor was implemented through a transformation of the existing UML Profile in the company to a corresponding Ecore model and using Xtext to generate the textual grammar. Two more transformations were generated using higher order transformations. These transformations allowed switching from the graphical model to the textual model and vice versa. With regards to

	Purposed	Cost-Effective	Configurable	Trusted	Scalable	Portable	Valued	Ubiquitous
Paper A		✓				✓		✓
Paper B		✓		✓	✓	✓	✓	
Paper C		✓	✓	✓				
Paper D			✓		✓			

Table 1.2: Research Contributions

this approach, we identified seven main challenges which are:

C1: Storage and Versioning of models – When having both graphical and textual models, which models should be persisted in the repository in a way that they are accessible in both notations?

C2: Synchronization of models – How can both models be in sync to ensure that they are not evolving separately?

C3: Layout and Formatting of models – How can one switch between the two notations and keep the original layout (for graphical models) and formatting (for textual models)?

C4: Model references – How can links between models be preserved and maintained as developers switch between different notations?

C5: Minimal DSL – Since the UML metamodel is huge and therefore hard to maintain as the DSL evolves, how can a minimal set of classes that are used be identified and maintained?

C6: Names in Model Elements – How can graphical elements that have no names but rather XMI IDs be converted to textual editors that require readable names?

Analyzing the above challenges we can see that apart from the tool-specific challenges such as C3, C5, and C6, the rest of the challenges are actually traceability challenges. For instance, C2 is on how to ensure that the graphical and textual models do not evolve separately and our proposal for this to use the transformations to switch between one model to the other. The transformation rules contain mappings between elements in the textual notation and elements in the graphical notation which act as implicit traceability links. In other words having transformations that are executed every time a model is saved can ensure consistency between the models without the users having to do anything. This is towards making traceability *ubiquitous* and *cost-effective*. C5 is also a traceability challenge on how to maintain the links between different models when the developers switch between the different notations. This is in connection to making traceability *portable*. There should be a way to transfer the inter-model links (traceability links in this context) from the graphical notation to the textual notation and vice versa. Identification of these challenges led to widening the scope of the next studies to traceability in general rather than tool integration.

1.8.2 Paper B: Persisting Software Traceability Challenges in the Automotive Domain

While we are aware of the grand challenges of traceability that have already been identified, we wanted to understand how the challenges reported in literature compare to the challenges currently found in practice, especially in the context of embedded systems development. Since embedded system development is a wide scope, we narrowed down our study to first only study traceability challenges for embedded systems developed in the automotive domain. To achieve this, we conducted a tertiary literature review on existing traceability literature. We examined 20 papers and extracted a list of challenges and solutions they report. We then conducted a case study with an automotive supplier where we used interviews and observations to collect data on their traceability processes and challenges. We interviewed a total of seven participants from different roles including developers, architects and traceability experts. The participants were also from two different departments each developing embedded systems. This allowed for data triangulation. We compared the challenges and solutions from literature with those we found at the company.

Our study revealed the challenges that have been solved, partially solved or remain unsolved. Our interest is in the unsolved challenges as it is where further research can be done. The unsolved challenges are as follows:

C1: Manual work – For traceability to be established a lot of manual work is needed in creating and maintaining the links. This is expensive and error prone.

C2: Traceability is perceived as an overhead – Developers need to create and maintain traceability links on top or their daily work. This is perceived as an overhead since the creators and users of traceability links are different groups of people.

C3: Lack of visualization tools – When traceability has been established, especially in large projects, the resulting traceability network becomes large. The users of traceability need visualization tools to be able to comprehend and use the links.

C4: Difficulty of assessing traceability – There is a lack of methods and tools for assessing the quality of the maintained traceability. When methods are available they are manual and error prone.

C5: Difficulty to measure return on investment – for each investment one expects to gain a return. Since traceability benefits are not visible immediately but over time, it is difficult to evaluate these benefits.

C6: Lack of universal standards – Sometimes traceability needs to be exchanged between departments and even between development organizations. Without a standard on how the links should be created and stored, exchanging traceability is difficult.

With respect to the grand challenges, this paper gives a detailed description of the challenges of making traceability *cost-effective* (C1, C5), *trusted* (C4), *scalable* (C3) and *portable* (C6). It describes how the challenges are experienced

in practice with respect to the automotive domain and why some of the existing solutions in literature do not work in this context.

1.8.3 Paper C: Traceability Maintenance: Factors and Guidelines

From our previous study (Paper B), we know that one of the persisting challenges is, establishing traceability is expensive as it requires manual work for both the creation and maintenance of the links. While a lot of effort is invested in companies to create the links, if the links are not updated as the artifacts they connect evolve, the links become outdated and thus useless. It is therefore as important to put effort in maintaining the links as in creating them. Creation of the links is harder to automate or semi-automate because the initial set of links does not exist and needs to be created from scratch. Maintenance on the other hand is comparably easier to automate or semi-automate because the links and information on the artifacts they connect already exist. However, the extent to which tool support for traceability maintenance can be provided varies depending on a number of factors. It is therefore important for the development organizations (when selecting traceability tools) and tool vendors (when developing traceability tools) to understand these factors.

We conducted a study to identify the important factors that affect the extent to which a traceability tool can provide maintenance support. We analyzed data from two sources: two focus group sessions that were aimed to elicit traceability requirements for a traceability solution and 24 semi-structured interviews from a study aimed at investigating collaborative traceability management practices. We formulated guidelines from each of these factors and validated our guidelines with interviews of expert users and developers of industry used traceability tools.

Our results identified the following four influential factors of traceability maintenance:

F1: Versioning – traceability maintenance is affected by whether or not a Version Control System (VCS) is used. The granularity of the changes that can be extracted from a VCS influences the extent to which traceability maintenance can be supported. From this factor, we suggest the following guidelines: G1) Version your traceability model just like all other artifacts and G2) Ensure that you are able to extract explicit deltas for all models from your chosen VCS.

F2: Tool Boundaries – In a practical scenario, it is expected that multiple tools are used to manage the different development artifacts. Maintaining traceability between artifacts in different tools is difficult as the tools are not connected to each other in any way. We elicited three tool-boundaries scenarios that can be possible. One scenario is where a holistic tool is used to support manipulation of all artifacts. While traceability becomes relatively easier in this case, in practice finding a holistic tool that supports all development activities is difficult. The second scenario is having a dedicated traceability management tool to connect the different tools. While this is feasible, it requires the tool is able to connect to the existing tools in the development tool chain. This can be done through

tool adapters. However, for each new tool, an adapter needs to be created which is not a trivial task. A third scenarios is a mixed scenario where a traceability tool is combined with tools for other functions e.g., requirements management tools. This scenario inherits all the negative aspects from the other two scenarios. From this factor we derive the following guidelines: G3) Traceability tools should provide well defined interfaces and easy, direct access to managed traceability links; G4) Aim for either a holistic solution or a completely separate traceability tool with a carefully designed tool adapter concept, avoid combining strategies and G5) Use a common standard as "glue" to simplify the development of adapters.

F3: Configurable Semantics – The degree to which a traceability metamodel/information model captures domain-specific semantics and whether the links are explicit or implicit influences how the consistency of traceability can be defined. This is because consistency must be defined and tailored to specific domains. For instance, ensuring that a traceability link from a requirement to a test is truly linking a requirement and a test, one needs to be able to define the semantics of the concept of a requirement and a test within the link in the traceability metamodel. From this factor we derive the following guidelines: G6) Avoid implicit, convention-based traceability links and strive instead for explicit links that can be checked with tool support and G7) Prefer domain-specific, semantically rich traceability metamodel as this simplifies traceability maintenance.

F4: Consistency Specification – The solution space for consistency specification spans two dimensions: the process by which traceability is maintained and the type of functions that are used to maintain traceability. From the process dimension, there are two ends of the spectrum: 1) traceability is maintained in a top-down approach and inconsistencies in traceability links are fixed immediately and 2) is a bottom-up and ad-hoc process of maintaining the traceability links where the links are fixed on demand. From this dimension, we derive the following guideline: G8) Ensure that your TM solution supports a flexible combination of both top-down and bottom-up maintenance approaches.

The other dimension of consistency specification is the type of consistency function specified. This can be manual, semi-automatic or automatic. For this dimension, we derive the following guidelines: G9) Support an integrated mix of manual and complementary automated approaches to consistency specification and G10) For automatically generated links, prefer no links at all over (possibly) inconsistent links.

The factors we identified and the guidelines we formulated are aimed towards making traceability more *configurable* (G3, G4,G5, G7, G8, G9) and *trusted* (G1, G2, G6, G10). The combination of all the guidelines aims to solve the traceability maintenance problem by reducing the manual work of maintaining the links and therefore supporting a *cost-effective* way of managing traceability links.

1.8.4 Paper D: Capra: A Configurable and Extendable Traceability Management Tool

From the previous studies, we gathered knowledge on the challenges of traceability and also specific requirements for a traceability solution. Analyzing the requirements, we discovered that there is no one solution that could cover all the requirements as they greatly varied depending on the development organization. In some cases some of these requirements were contradictory, for instance, where the links should be stored and which links should be in the metamodel. With these requirements in mind we developed a traceability solution that can be customized and extended to support the diversity in the requirements. This was done using design science where we gathered requirements from both our industrial and academia partners, implemented a prototype and did demonstrations to validate the prototype. From these demonstrations we got feedback and more requirements for another iteration. The tool was developed starting in September 2015 and is still being actively developed. Since August 2016, the tool is an Eclipse open source project[7] making it available to a larger community.

In our solution, we followed the concepts described in Paper C to ensure that the tool will support maintenance as much as possible but also as another way to validate some of the guidelines we proposed. As a result, the tool is extendable in three different perspectives:

The traceability link types – It is possible to create link types with different semantics depending on the needs of the development organization. The tool allows for the links to be defined in a traceability metamodel and this can be extended with domain-specific links, thus following guideline G7.

The artifacts that can be traced – Since different development organizations use different tools one cannot develop a traceability tool that is complete. There are always going to be more tools that are not supported. To overcome this problem, our tool is a dedicated traceability management tool and therefore follows G4. It allows for tool adapters to support different tools to be added on a need to need basis. Therefore it is extendable. The tool uses EMF [27] as a common standard for the adapters which follows guideline G5. As adding adapters requires some technical expertise, the tool ships with adapters to support common a languages such as EMF and UML, programming languages such as Java, C/C++ and PHP and general purpose tools such as MS Office. Adapters for more specific formats such as ReqIF and task tickets from common task management tools such as JIRA and Bugzilla are also available. Since the tool is also an open source tool, more adapters can be contributed by the open source community. As the adapters can be turned on and off, the tool is both configurable and scalable.

The storage of the artifacts – The needs for storage of traceability links can differ depending on the development organization. For instance, in some cases the participants in our study preferred to store the links per project,

[7] `https://projects.eclipse.org/projects/modeling.capra`

while other participants preferred to store the links per workspace and others per repository. Therefore, the tool allows this to be configured. This makes it possible for the tool to adhere to G1 and G2 depending on the storage and VCS choice of the end users.

Following guideline G3, the tool also provides interfaces to expose the traceability model to various tools that, e.g., provide visualization. To show the feasibility of this, Capra currently has two different visualization options, one utilizing PlantUML [45] and one using the Eclipse Graphical Editing Framework (GEF) [46].

The development of the tool therefore aims to address challenges of achieving *configurable* and *scalable* traceability.

1.9 Conclusions

In this section we discuss how we addressed our research goals and how the research questions were answered by the studies appended in Chapter 2 to Chapter 5. The first goal of the thesis was to investigate current traceability challenges in the development of embedded systems. As discussed in Section 1.5 and 1.6, our research first started with a tool integration problem which is only a subset of the traceability topic in general. The first research question (RQ1) was therefore; how can editors be integrated in a development environment in a way that is cost-efficient and avoids inconsistencies as models evolve? To solve this problem we implemented a model transformation from the metamodel of the graphical model to the metamodel of the textual model. Using the textual metamodel we implemented a textual concrete syntax that could be used to create models in text. To ensure that users can switch between graphical and textual views, we implemented two transformations that facilitate transforming from graphical to textual models and vice versa. In order to make sure that the same version of the graphical model is consistent with the corresponding textual model, the transformations were executed every time a model is saved. To make sure that our solution is cost-effective, we implemented Higher Order Transformations (HOT) which generate other transformations. This way, in case of change to the initial graphical metamodel, the transformations that allow switching from graphical to textual models do not have to be manually written and can be generated instead. To summarize, in this study, model transformations were enablers for both consistency maintenance and traceability between models. We also uncovered traceability challenges that manifest in practice in such model-driven developments scenarios, for instance, how to ensure that links can be established between the different notations and how to keep all models synchronized.

Further addressing the first goal, we conducted a tertiary literature review and a case study to identify traceability challenges from literature and also investigate how they are experienced in reality. Our aim was to answer RQ2 (What are the current traceability challenges for embedded systems development?). From this study we identified that creation and maintenance of traceability is done manually which makes the process expensive and error prone. It also makes the engineers who create these links feel like it is an unnecessary overhead. When traceability is established, we identified that

there is a lack of methods to asses the quality of the links. The existing techniques are still manual and error prone. Another challenge is the lack of visualization tools which makes it hard to use the traceability links. Also when the development of the system is done across organizational boundaries, the lack of universal standards for traceability makes it difficult for the links to be shared and exchanged. The last challenge identified is the difficulty of measuring the return on investment of traceability.

With RQ1 and RQ2 answered, we decided to first look into the challenges that are tool related. This was to address Goal 2 (Propose conceptual tool solutions that can solve the identified challenges). We decided to investigate the manual work invested in maintaining the links and how this work can be minimized through tool support. We therefore investigated RQ3 (What are the primary factors that affect and to what extent a traceability management solution can provide traceability maintenance?). We identified four important factors which are: 1) versioning, 2) tool boundaries 3) configurable semantics and 4) consistency specification. From these factors we also formulated ten guidelines that can be used by traceability tool developers when developing traceability solutions to ensure that their solutions can provide traceability maintenance support.

To address Goal 3 (Develop prototypes that demonstrate how traceability solutions can be implemented), we used results from Goal 2 as input and conducted a study using design science. In this study, we collected requirements for a traceability solution and elicited differing requirements from several development organizations. We therefore investigated RQ4 (How can a traceability tool be implemented in such a way that it is configurable and extendable?). Our study revealed that a configurable and extendable traceability management tool needs to be flexible in three ways: 1) allow for custom traceability metamodels (traceability information models) to be defined by the development organization, 2) provide mechanisms for additional artifacts types to be supported and 3) allow the storage of the traceability links to be determined by the development organization. To demonstrate the feasibility of this, we implemented Capra[8] which is a prototype traceability solution. This is only an initial step for our research and in the near future we plan to investigate how the tool can be used in a practical environment and how many of the challenges identified it will address.

1.10 Future Work

As mentioned in at the start of this chapter, the overall goal of the research is *to improve traceability processes and tools for embedded systems development.* In this thesis, we took first steps towards this goal by identifying current challenges, proposing conceptual tool solutions and implementing a prototype to demonstrate the feasibility of the proposed solutions. To be able to reach our goal we also need to investigate traceability processes (comprising of the different traceability activities such as creation, maintenance and use) and how these processes can be integrated in existing development environments to ensure that traceability is established and maintained in a cost-effective

[8] `https://projects.eclipse.org/projects/modeling.capra`

manner. Concrete steps that we will take to achieve our goal are as follows:

Firstly, since we already have a prototype tool, we intend to use it and conduct case studies with development organizations in order to investigate how the tool will be used, how it fits in their processes and how we can improve the current solution. We plan to conduct such studies with development organizations from different domains to be able to compare their processes and how traceability fits in their environments.

Secondly, we already have knowledge that the traceability metamodel can differ from development organization to development organization and even from project to project. We therefore aim to collect traceability metamodels according to the domains and development environments in which they are used. The idea is to come up with a catalog of traceability metamodels mapped to domains and development environments. This catalog can act as a starting point for practitioners who want to establish traceability in their companies.

Lastly, one of the reasons companies implement traceability is because of the standards that they have to follow. This applies to safety-critical domains such as automotive in which development organizations have follow ISO 26262 [47], the medical domain in which development organizations have to follow the IEC 62304 safety standard [48] and the avionics domain where development organizations have to follow DO-178B/ED-12B [49]. Other domains such as avionics also have their standards. As part of our future work, we plan to investigate these standards in order to identify traceability requirements that they impose. This will enable us to compare the standards from a traceability perspective and also give us insights on further requirements that can be used to tailor traceability processes, metamodels and tools for use in the different domains.

Chapter 2

Paper A

On Integrating Graphical and Textual Editors for a UML Profile Based Domain Specific Language: An Industrial Experience

S. Maro, J.-P. Steghöfer, A. Anjorin, M. Tichy, L. Gelin

13th International Conference on Software Language Engineering(SLE 2015), Pittsburg, USA, October 23-27, 2015.

Abstract

Domain Specific Languages (DSLs) are an established means of reducing the gap between problem and solution domains. DSLs increase productivity and improve quality as they can be tailored to exactly fit the needs of the problem to be solved. A DSL can have multiple notations including textual and graphical notations. In some cases, one of these notations for a DSL is enough but there are many cases where a single notation does not suffice and there is a demand to support multiple notations for the same DSL. UML profile is one of several approaches used to define a DSL, however most UML tools only come with graphical editors. In this paper, we present our approach and industrial experience on integrating textual and graphical editors for a UML profile-based DSL. This work was conducted as part of an explorative study at Ericsson. The main aim of the study was to investigate how to introduce a textual editor to an already existing UML profile-based DSL in an Eclipse environment. We report on the challenges of integrating textual and graphical editors for UML profile-based DSLs in practice, our chosen approach, specific constraints and requirements of the study.

2.1 Introduction

The term Domain Specific Language (DSL) refers to a language that is created for specific tasks [50]. A DSL captures design patterns that are common in a particular domain. Compared to General Purpose Languages (GPLs), a DSL focuses on the patterns that are used in a specific domain therefore avoiding all the general notations that are not needed within the domain. This makes creation of applications easier and designers can write less code (as little as 2%) than when using a GPL [51]. Applications developed using DSLs tend to be more concise, easier to maintain and reason about and above all can be developed quickly [52].

At Ericsson, DSLs are used to raise the abstraction level with which engineers create applications. This gives the engineers an opportunity to focus on the problem at hand and not worry about implementation details. Moreover, the fact that software at Ericsson is usually deployed on various hardware platforms is another main reason for the adoption of DSLs. It would be inefficient for engineers to write different code for every hardware platform. With DSLs they can reuse the logic models which are platform independent and this proves to be more efficient.

A DSL can have a textual notation or a graphical notation. When a DSL is large, covers a wide aspect and has different types of users like Ericsson's DSL, having one notation often does not suit the needs of all its users. This is because some cases are easier to specify when using a graphical notation while others can be conveniently specified using a textual notation [44]. On the one hand, a graphical notation for modelling applications has advantages like reducing the chances of errors, providing visualization and hence easing understanding of the system being created. On the other hand, using text based modelling has advantages such as speed of creation and editing, speed of formatting and a wide availability of editors [44].

Unfortunately, when opting to use UML profile-based DSLs, only graphical editors are provided by most out-of-the-box UML tools. At the moment, Ericsson has a UML profile-based DSL for modelling applications for baseband switches. The company is using the Rational Software Architect (RSA) tool from IBM which only provides a graphical editor for UML models. Based on the cases that they model, some engineers have requested a textual editor to be provided because it would simplify their modelling tasks. At the same time, these engineers have to work and share models with other engineers who are using the graphical editor. Therefore, the company is investigating the possibility of adding a textual editor for the existing DSL.

Having a DSL with both textual and graphical editors being used simultaneously raises several problems. One of the main problems is how to maintain the two editors (textual and graphical) without adding substantial maintenance effort to the company. This means that when the DSL needs to be updated one should not have to do a lot of manual work. Another problem that arises is how to avoid information loss when users switch between graphical and textual views.

The main contribution of this paper is to present a practical approach on how to semi-automatically obtain a textual editor for a UML profile-based DSL and how to enable users to switch from one view of a model to another, i.e.,

to switch from the graphical view to the textual view and vice-versa without losing any information. Since the study was carried out in industry, we present the challenges that arise when having both textual and graphical editors for the same DSL.

The rest of the paper is organized as follows. Section 2 gives an overview of the DSL at Ericsson and Section 3 outlines the challenges of having textual and graphical editors for the same DSL. Section 4 describes our approach while Section 5 gives an evaluation of our tooling. Section 6 is a discussion on the approach in relation to the challenges. The paper concludes with Section 7 that discusses related work and Section 8, which gives a conclusion and proposes future work.

2.2 Industrial Case

Ericsson uses an in-house developed DSL to create applications for its baseband switches. This DSL is built using UML and UML profiles. Their UML profile is known as Hive profile and is divided into three major parts: Behaviour, Structure and Deployment. These serve different purposes when it comes to creating applications. The Hive DSL in total is made up of 17 stereotypes, 1 Enumeration and 1 class from the Hive Behaviour profile, 2 stereotypes from the Hive Structure profile and 14 stereotypes from the Hive Deployment profile.

At Ericsson, developers create models of applications using this DSL and later use model transformation tools to transform the models into C code (.c and .h files), which can be compiled into working applications. The transformations are done in three steps:

[a] A Hive instance model is subjected to a model to model transformation that transforms it to a Dive model. Dive is another Ericsson in-house DSL which is XML-based. The Dive model is never manually edited by the user but always has to come from a corresponding Hive model using this model to model transformation.

[b] From the Dive model, another model to model transformation is done that transforms the Dive model into a C instance model that conforms to the C Abstract Syntax Tree (AST).

[c] From the C AST, a model to text transformation is performed to get .c and .h text files which can be compiled into working applications.

Figure 1 illustrates the above three steps.

The company uses RSA from IBM as their main editor which only provides a graphical editor for creating and editing UML models. As this does not suit the needs of all users, the company wants to add a textual editor so that their developers can have the option of modelling using text or graphics. For this the company has the following requirements:

[a] Add a textual editor for the already existing DSL.

[b] The company should not have to maintain the graphical and textual editors separately.

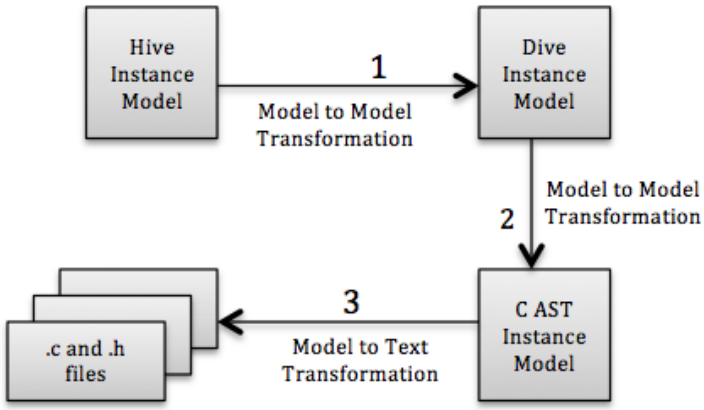

Figure 2.1: Transformation Tool Chain

[c] The engineers should be able to switch between textual and graphical views without losing any information.

One of the cases that we investigated at the company was how to model the behaviour part of the Hive profile using text. To model behaviour, the company uses activity diagrams extended with the Hive Behaviour profile. An activity diagram in UML is a behaviour diagram which shows flow of control or object flow with emphasis on the sequence and conditions of the flow [53]. Figure 2 shows an extract of the Hive Behaviour profile with two stereotypes: `HiveAction` and `HiveVectorAction`. Both the stereotypes have two tagged values which are `taskPriority` and `operation`. Both stereotypes extend the `CallOperationAction` metaclass from the UML metamodel.

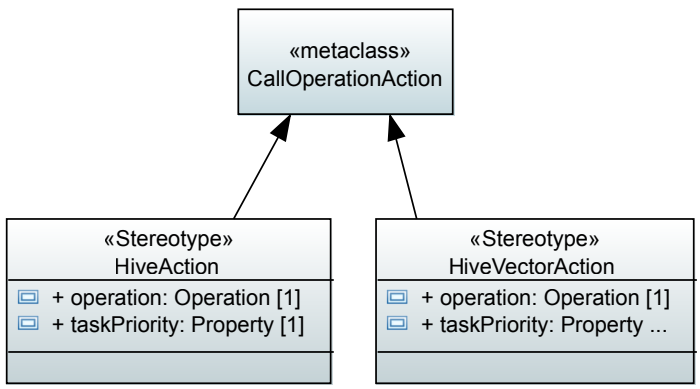

Figure 2.2: Extract from the Hive Behaviour Profile

2.3 Challenges

This section describes the challenges that we identified during the study. These are mainly general challenges related to having a textual and graphical editor used in a working environment and some specific challenges related to integrating a UML profile-based DSL with Xtext.

2.3.1 Storage and Versioning of Models in Repositories

In many companies it is common that more than one designer works on the same model and this is facilitated by using collaboration and versioning tools such as Git [54]. When some designers can edit the same model in text and some edit it graphically, the choice of which format (textual or graphical) to store in the repository is not a trivial one. Storing the models as graphical models means putting in place tools that can help designers to version, merge and resolve conflicts on a graphical model level. On the other hand storing them in text means using tools for versioning, merging and resolving conflicts that are on a textual level. Either way, deciding to store one version means that designers working in the other version need to first convert their models before committing their changes. This leads to other challenges such as synchronization and issues of maintaining the layout, which are discussed next. Storing both versions of the model is also an option that can be considered, but this means that one has to make sure that both models (textual and graphical) can be versioned and synchronised.

2.3.2 Synchronization of Models

When a user is editing a model in graphical format and decides to switch to textual notation, both the textual and graphical models should be in sync. This means that the tool should be aware of which text model represents which graphical model. This is a challenge if the graphical and textual views are not linked to each other as the source and target models can evolve separately and become out of sync. There is therefore a need to be able to link a transformed model to its source model in order to keep them synchronized.

2.3.3 Graphical Layout of the Model and Pretty Printing

When a user creates models in graphical format, they can arrange/format the diagram in a way that is suitable for them. For instance they can make the icons bigger, move the model elements to certain positions or arrange them in a particular way. When these models are transformed to textual models and then back to graphical models the user expects the layout of the original graphical model to be maintained. This is a challenge as the textual model does not store any information about the graphical layout of the model elements unless additional measures to facilitate this are put in place. As models get bigger, the layout that a user has modified becomes important and the auto-layout provided by the tool does not suit the user. Moreover, when a designer styles his or her text in a certain format using the text editor and then transforms the file to a graphical model and then back to text, the custom text formatting is also lost.

2.3.4 Model References

It is common that developers create several models that are linked to each other instead of creating one big model. This means that most models have model references to other models. This is a problem, as when switching from one view to another, one has to decide on whether to switch only the current model that is being worked on or also switch all models linked to the current model. Switching only the model being worked on means that we end up with a model containing a mixed grammar, i.e., a graphical model with links to a text model or vice versa. Switching the current worked on model, and all its referenced models may result to a bunch of models being transformed, which is not efficient when having many linked models.

2.3.5 Minimal DSL

The UML metamodel is a very huge one. For instance UML 2.0 has 265 model elements and 763 relationships [55]. This makes it quite complex to work with and in many cases, companies only use a small part of this metamodel. In order to come up with a small concise metamodel representing the UML profile, it is necessary to identify which parts of the metamodel the profile uses and which other classes the company uses from the UML metamodel. This is a difficult task because in many cases it is not clear which classes are actually used by the company and also because the UML metamodel contains many elements that are connected to each other and sometimes these connections are implicit.

2.3.6 Names in Model Elements

This challenge is mainly related to UML. The EMF implementation of UML uses special IDs called XMI IDs to identify model elements. This means that when a user creates an element in UML this element is assigned an ID by default. Even when a user does not give the element a name, UML can still identify that element with the ID it has. This has led to a habit that many designers do not give names to UML elements unless the names are necessary to them. For instance when modelling activity diagrams, most of the designers give names to the actions but not to the fork nodes or join nodes. However, when it comes to EMF textual editors, all elements are identified by using qualified names. If a designer creates a model element in text, the editor does not assign any ID to the element, it is the designer that needs to give unique names to the elements. This is a challenge as when transforming UML models to textual models, the produced text model elements are also unnamed. In text models we cannot have a reference to an element with no name so the resulting model breaks.

The main challenge here is that introducing the textual editor means that either the designers have to start giving names to all model elements in their models or the names should be automatically generated for elements that have no names when transforming to text models. Generating names also implies that when a user switches from text to graphics and back, the graphical model will have the generated names added. The XMI IDs from the UML models could also be considered for use as unique model element names in text but these have a format that is not user friendly.

2.3.7 Inconsistent Models

An inconsistent model is a model that does not adhere to the constraints of the metamodel, it has errors. When switching from a graphical model to a text model, if the transformation produces an inconsistent model then this model cannot be serialized in the Xtext syntax. The designer therefore needs to make sure that when transforming a graphical model, this model should not lead to an inconsistent text model. For example in the Hive case, the join node in textual metamodel has a constraint that it can have several inputs and only one output. So if a designer tries to transform a join node that is not connected to any input or any output then the result of this transformation cannot be serialized in Xtext textual syntax. The error must be fixed first for the serialization to work.

2.4 Approach

This section describes the approach that we used to address the problem of integrating textual and graphical editors in an Eclipse environment. It should be noted that the approach described here has been taken due to the fact that the company was already using a UML profile-based DSL. The approach also aims to address some of the challenges discussed in the previous section. This section is divided into two parts, the first part describes how to obtain the textual editor and the second part describes how to switch between graphical and textual views.

2.4.1 Obtaining the Text Editor

There are several EMF based plugins that can be used to generate textual editors with little effort. Since the company is already using RSA, which is also EMF based, going for one of these text editor generator plugins is a good solution. However, all these plugins need an Ecore model behind them in order to generate the textual editor. This Ecore model can be manually written but this means that every time the profile evolves, the Ecore model would need to be changed manually. To avoid this, we transformed the UML profile-based DSL into an Ecore model. The idea being that once the profile evolves, the Ecore model will be derived automatically.

The Ecore model obtained from the transformation could now be used with one of the text editor generator plugins to generate a textual grammar and editor for the DSL. For our case we used Xtext [56] to generate the grammar and the textual editor.

EMF comes with a functionality that can export UML models into Ecore models. Using this functionality was the first approach to transform the Hive profile to an Ecore metamodel for a DSL. This functionality worked well but it had one huge drawback since it also exports the whole UML metamodel to the exported Ecore model. This gives a very huge DSL with a lot of entities that are not needed. To overcome this we wrote our own ATL transformation and used a UML subset in our transformation instead of the whole UML metamodel.

To be able to obtain the UML subset, we need to know exactly which metaclasses are used. These metaclasses include those extended by the stereotypes

in the profile and also those that are used without any stereotypes. For some DSLs this set of metaclasses is known and for some DSLs it may not be so obvious which metaclasses are used. This is especially true when users use part of UML that is not extended by any stereotype in the profile. In such cases this list of classes that are needed can be obtained by running a transformation that takes an instance model of the DSL and returns a collection of all UML metaclasses used on that instance model. This will give a correct list of classes needed only if the instance models cover 100% of the DSL.

In case no such instance models exist, one can identify the needed UML metaclasses manually and create a list of these classes either as an Ecore or as another UML profile that will only be used to identify these metaclasses. Once these classes have been identified a transformation can be written that copies only these classes from UML to create a subset UML metamodel. This UML subset can also be created manually as an Ecore model that contains all the classes of the subset and their attributes. However if the DSL changes frequently then this subset can be hard to maintain. For the case of Ericsson, we used a manual approach to get a list of classes used from the UML metamodel. This was done by examining various existing models and identifying classes that were implicitly used from the UML metamodel. This worked as a solution because their profile does not evolve frequently.

Since we created the UML subset manually, we had the flexibility to get rid of model elements that we considered unecessary in our final textual language or add some model elements. However this change needs to be noted so that when transforming back to UML models, we know how to re-create UML model elements from the changes made. The mapping for this are stored in a trace model [57] because the relationship of UML and the Ecore model will no longer be a direct one to one relationship. A trace model is a model that defines the relationship between a source model and a target model. In our case since one of the diagrams we modelled in text is activity diagrams, instead of having controlflows with source nodes and target nodes elements we added an attribute called "dependsOn" to activity nodes. This was done to make modelling of the flow from one node to another easier in text.

We wrote an ATL transformation that takes the UML profile and UML subset as input and produces an Ecore model as output. We used the produced Ecore model from the transformation as input to the Xtext plugin to generate the grammar and textual editor. This transformation could also be written using any transformation language. The mappings used to convert UML to Ecore follow the ones used in the UML to Ecore eclipse plugin [58] and also according to the relationship between UML and Ecore as described in [59]. The mappings are summarized in Table 1.

Figure 3 shows an extract of the Hive profile, part of the subset needed and the resulting Ecore model that was obtained from the transformation. In the figure, a stereotype called `HiveAction` is transformed to an EClass called `HiveAction`. The property of the `HiveAction` stereotype called `operation` which has a type of the UML Operation metaclass is transformed to an EReference called `operation` whose type is also an EClass called `Operation`. This `Operation` EClass comes from the `Operation` metaclass in the UML subset. The UML extension relationship that is represented by the property named `base_CallOperationAction` of type `CallOperationAction`, is also

Table 2.1: UML to Ecore Mappings

UML	Ecore
Profile	EPackage
Stereotype	EClass
Metaclass	EClass
Property(Primitive Type)	EAttribute
Property	EReference
DataType	EClass
Enumeration	EEnum

transformed to an EReference named `base_CallOperationAction` of type `CallOperationAction`. Similarly, the EClass called `CallOperationAction`, comes from the UML subset. This is done for all the stereotypes and their properties. From the UML subset, all the classes and their attributes are copied to the Hive Ecore model. So a class named Activity in the UML subset is transformed to a class named Activity in the Hive Ecore model.

Figure 2.3: Hive Profile to Hive Ecore Metamodel

The grammar generated by Xtext from the Ecore metamodel of the DSL was not very usable because it had a lot of unnecessary syntactical structures and all the enumerations literals were missing. This is an Xtext specific issue for grammars that are automaticaly generated. Therefore, we manually edited the grammar in order to come up with something that is actually readable and usable.

Figure 4 shows our approach to obtain the textual editor from the Hive profile. It also shows which parts have to be done manually every time the Hive profile evolves and which parts have to be done only once.

With the setting shown in Figure 4, when the profile evolves, the following needs to be done to update the editor. First, one needs to check if the profile is using a metaclass that is not included in the UML subset and add all missing metaclasses and their attributes. Then the new profile and the UML subset will be used as input to the already existing ATL transformation to produce the new Hive metamodel. To update the grammar there are two choices, the first choice is to re-generate the whole grammar and the editor, but this means also re-doing the entire manual editing that was done to the grammar before. The second option is to replace the old Hive metamodel with the new one, this way when the editor is compiled, the changes will be detected and one can update the grammar to match the new Hive metamodel manually.

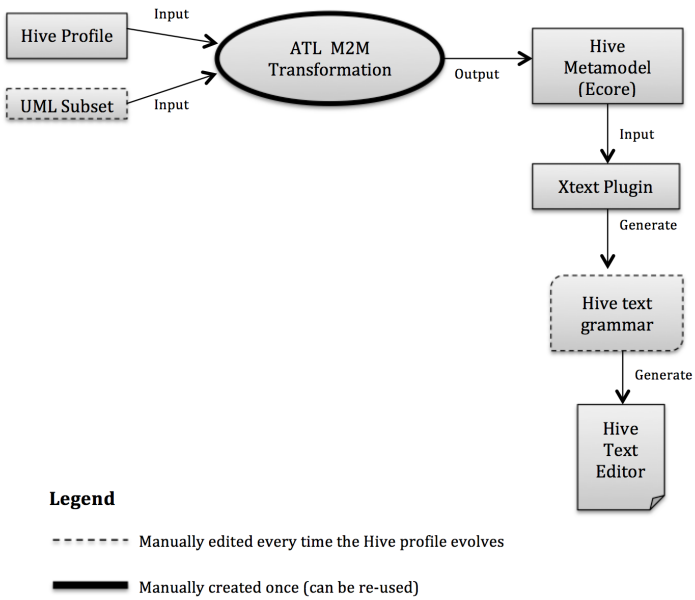

Figure 2.4: Generating the textual editor from the Hive profile

2.4.2 Switching between Graphical and Textual Views

Once the text editor was in place, it was now possible to create models using text. To switch from Xtext to UML and vice versa, two transformations are required: one is to transform graphical models to text models and the other is to transform text models to graphical models. Xtext also provides a mechanism to obtain an XMI version of the model written in text. This way the model could be used as an input to a transformation so that it can be transformed to its UML version. Figure 5 shows an example of a UML model in graphical format and Listing 1 shows its corresponding text model.

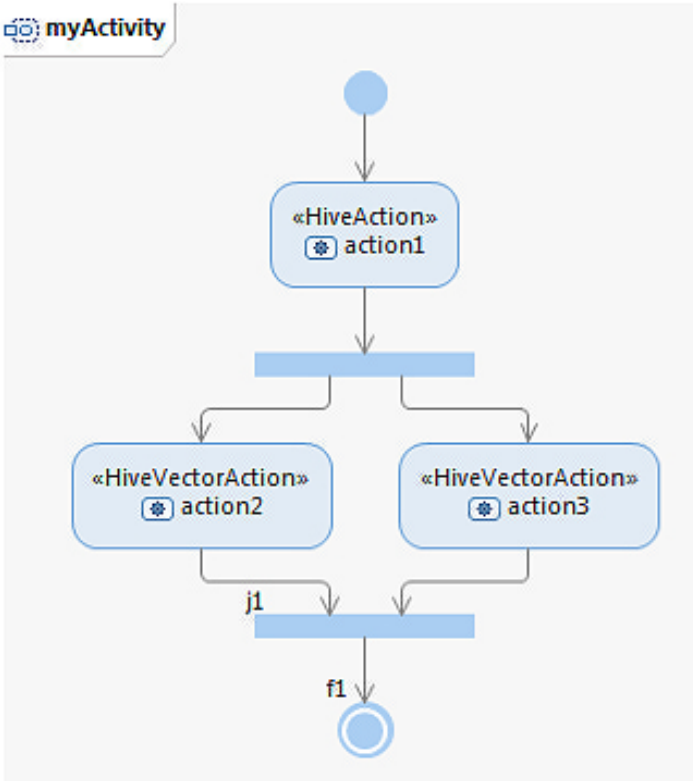

Figure 2.5: Graphical view of an activity diagram.

```
Activity myActivity {
 nodes{
   Initial node init
   CallOperationAction action1 dependsOn init testOperation
    HiveAction testVariable
   CallOperationAction action2 dependsOn action1 testOperation
    HiveVectorAction testVariable
   CallOperationAction action3 dependsOn action1 testOperation
    HiveVectorAction testVariable
   MergeNode m1 merges (action2,action3)
   ActivityFinalNode f1 dependsOn m1
 }
}
```

Listing 2.1: Textual view of the activity diagram shown in Figure 5.

Because we did not want to have any effort applied to these transformations when the Hive profile evolves, we used ATL HOT to generate these transformations instead of writing them manually.

Keeping in mind that the metamodel that was used for the Xtext language is generated from the UML profile and UML subset, most of the information needed for our instance model transformations is available in these two models (UML profile and UML subset). We therefore wrote a HOT that takes the UML profile and the UML subset as input and produces instance model

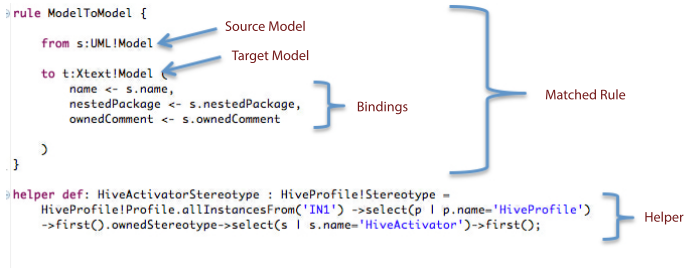

Figure 2.6: Structure of an ATL Transformation.

transformations as output.

Generally, an ATL transformation is composed of matched rules. Matched rules define what the source element is and what target element it should be transformed to. Also the rule contains bindings, which define how attributes of a source model element should be transformed into attributes of a target model element. ATL transformations also have helpers, which are functions that can be called within ATL rules. Generating an ATL transformation means generating the rules, bindings in the rules and helpers as well. The code snippet in Figure 6 shows an example of the structure of an ATL transformation.

To generate the matched rules of our transformations, we use information from the Hive profile and UML subset. For example, when generating a transformation that will transform a UML model to an Xtext model the following is done.

- From the Hive profile, each stereotype that is not abstract is transformed to an ATL matched rule. This matched rule will have one source which is an instance of the metaclass extended by the stereotype and two outputs, one corresponding to the extended metaclass and one corresponding to the stereotype itself.

  For example Listing 2, shows an example of a generated rule from a stereotype called `HiveMapToFunction` which extends a metaclass called `Transition`. This rule (Listing 2) will transform an instance of a UML Transition which has the stereotype `HiveMapToFunction` applied to it (Line 3-5 in Listing 2) to two model elements in the Ecore model. The first element is an instance of a class called `Transition` (line 7 in Listing 2) which corresponds to the UML metaclass and an instance of class called `HiveMapToFunction` (line 14 in Listing 2) which corresponds to the stereotype.

- Each stereotype that is not abstract is also transformed to a helper that identifies the stereotype by name from the Hive profile in the transformation. An example of a helper function generated is shown in listing 3 and this helper function is generated from a stereotype called `HiveMapToFunction` and is used to identify this stereotype from the profile. This helper function is called in the generated matched rule in line 4 of Listing 2.

- Properties in the stereotype are transformed into ATL bindings. Examples of these ATL bindings can be seen in line 15 to 19 of Listing 2).

For instance line 15 shows that the value stored in the property called `threadId` from the `HiveMapToFunction` stereotype will be stored in an attribute called `threadId` in the resulting Ecore model.

- Similarly, properties from the UML metaclasses are transformed into ATL bindings. Examples of these ATL bindings can be seen in line 8-11 of Listing 2.

  For instance line 8 shows that the value of the property named `kind` in UML will be stored in an attribute called `kind` in the resulting Ecore model.

```
1  rule HiveMapToFunctionStereotypedClass {
2   from
3    s : UML!Transition (
4     s.isStereotypeApplied(thisModule.
       HiveMapToFunctionStereotype)
5    )
6   to
7    t : XTEXT!Transition (
8     kind <- s.kind,
9     source <- s.source,
10    target <- s.target,
11    name <- s.name,
12    extension_HiveBaseMapToBehavior <- t1
13   ),
14   t1 : XTEXT!HiveMapToFunction (
15    threadId <- s.getValue(thisModule.
       HiveMapToFunctionStereotype, 'threadId'),
16    newTask <- s.getValue(thisModule.
       HiveMapToFunctionStereotype, 'newTask'),
17    taskPriority <- s.getValue(thisModule.
       HiveMapToFunctionStereotype, 'taskPriority'),
18    actionPackageFile <- s.getValue(thisModule.
       HiveMapToFunctionStereotype, 'actionPackageFile'),
19    actionPackageName <- s.getValue(thisModule.
       HiveMapToFunctionStereotype, 'actionPackageName')
20   )
21 }
```

Listing 2.2: Matched Rule created from a Stereotype

- For the metaclasses in the UML subset, each metaclass that is not abstract is transformed into an ATL matched rule. For example in listing 4, the UML metaclass named Class from the subset generated a rule called ClassToClass. (line 1 in listing 4). This generated rule is used to transform UML classes that are not extended by any stereotype. This constraint can be seen in line 4 of Listing 4.

- Attributes and references of the classes in the UML subset are all transformed into ATL bindings. This works well as long as there is a direct one to one relationship of the attributes from UML to Xtext and vice versa (line 10 to 13 of listing 4).

```
1 helper def: HiveMapToFunctionStereotype : PROFILE!Stereotype =
2   PROFILE!Profile.allInstancesFrom('IN1') ->select(p | p.name='
      HiveProfile')
```

```
  ->first().ownedStereotype->select(s | s.name='HiveMapToFunction
    ')->first();
\end{Code}

\begin{Code}[{ATL Matched rule generated from a Class in the UML
    subset.}]
rule ClassToClass {
  from s: UML!Class (
  s.getAppliedStereotypes() -> isEmpty()
  )
  to t:Xtext!Class (
    name <- s.name,
    ownedAttribute <- s.ownedAttribute,
    ownedOperation <- s.ownedOperation,
    ownedBehavior <- s.ownedBehavior
  )
}
```

Listing 2.3: ATL Helper generated from a Stereotype.

If the bindings from UML to Xtext do not have a direct one to one relationship (for example name to name), the HOT transformation needs more information in order to create these bindings. This extra information is related to the manual change that was made to the UML subset (discussed in Section 4.1) that affected the Hive metamodel (in Ecore). These mappings are stored in a trace model. The trace model we used is based on the ATL trace metamodel [57].

For example if we are transforming from a UML model to an Xtext model we know that the controlflow element in UML is not represented as a controlflow in text but as a `dependsOn` attribute. Therefore in our trace model, we create a trace rule called `ControlFlow` and add one link to it which has the source element as the source of the controlflow and the target element as the `dependsOn` attribute. Figure 7 shows an example of this trace model.

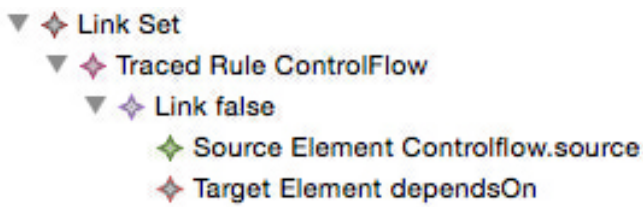

Figure 2.7: Example of a trace model.

Figure 8 summarizes how the two model transformations are obtained from HOT.

2.5 Evaluation

To evaluate our approach we applied it to two parts of the Hive DSL at Ericsson: the Hive Behaviour profile and the Hive Structure profile.

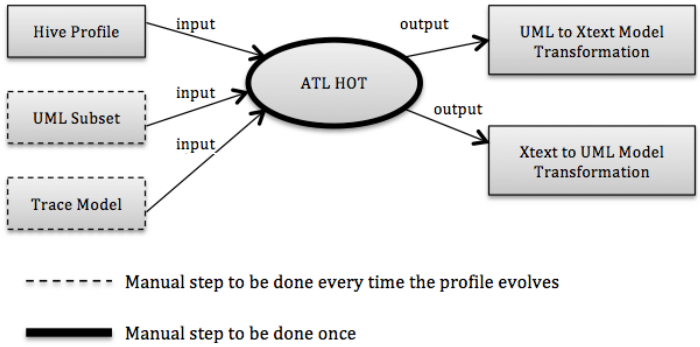

Figure 2.8: Generating Model Transformations.

The transformation from UML profile to an Ecore model with the help of a UML subset (as described in Section 4) worked for both the profiles. To further test the solution, we had to analyse if there is any information that gets lost or is added during the switch. To achieve this, we used three demo models that are available at Ericsson which are created using the Hive profile DSL. The demo models were provided as input to the transformation from UML to Xtext to obtain an Xtext model. We then converted the Xtext model back to UML using the transformation from Xtext to UML. The original UML model was compared with the one generated from Xtext using EMF Compare (see Figure 9). EMF Compare is an eclipse plugin that is used for comparison and merging of EMF models [60].

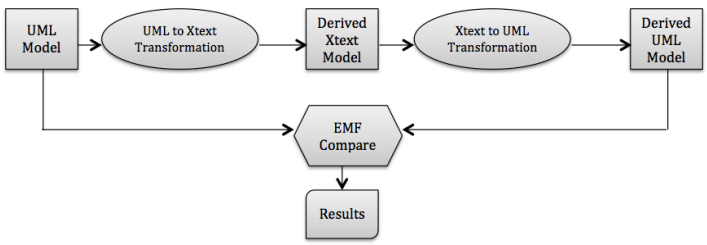

Figure 2.9: UML to Xtext evaluation process.

From the comparison done, we found out that even though the models were semantically equal, in some cases there were slight differences noted between the UML models and the ones obtained after switching to text and back to UML. On analyzing this we discovered that elements that had no names in UML were causing problems when switching to text. This has been discussed in details in Section 3.6. To overcome this we added an extra helper function to the transformations that generates arbitrary names for model elements in UML that were not named. This also implied that when switching back to UML from the generated Xtext model, these names also appeared in the UML model.

Another issue with our approach was the loss of graphical layout of the UML models. This is because we only transformed the semantic model and did not store the layout information anywhere. After a transformation the diagrams are lost and need to be re-generated. Regeneration uses the auto-layout that RSA provides. We did not implement a solution for this but propose that one can use incremental model transformations to solve the problem. Incremental model transformation is the kind of transformation where instead of the transformation creating a new target model every time, it checks which model elements are changed and updates only those elements in the target model. The target model is not recreated but rather updated with changes from the source model. Section 6.2 provides a further discussion on this.

Another comparison was made when switching from instance models created using the Xtext editor to UML and again back to Xtext. Since there were no models available in text, new ones were created for the purpose of this comparison. The comparison was made using the textual quick diff functionality in Eclipse (see figure 10). This is a functionality that lets a user compare text files side by side.

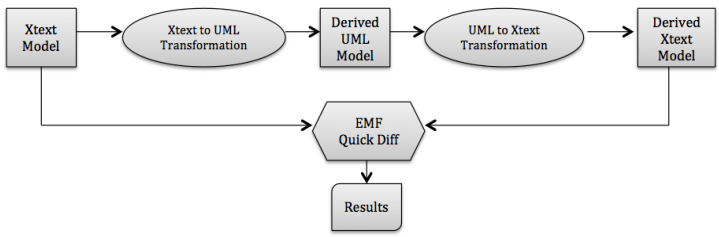

Figure 2.10: Xtext to UML evaluation process.

From this comparison, if a textual model had no comment inserted then the resulting model transformed to UML and back to Xtext was identical to the original text model. But when the text model had comments, these comments were lost and making the resulting model not identical to the original model during comparison. All the custom pretty printing that the user had made in the text is also lost after transforming to UML and back to Xtext. This challenge has been discussed in Section 3.3 and is due to the fact that the transformation created a new model every time it was run. Here we also suggest the use of incremental model transformations to solve this problem.

2.6 Discussion

This section discusses our approach in relation to the challenges that have been identified in Section 3.

2.6.1 Addressed Challenges

Our approach addresses four challenges so far: Challenge 2.3.2, 2.3.5, 2.3.6 and 2.3.7. Challenge 2.3.2 is about synchronization, how can we keep graphical and textual models in sync. With our current approach, for instance, when a

user is editing a model in one graphical view and decides to switch to textual view, all the information that was in graphical view is transformed to a textual version of the model. The vice versa is also true. This ensures that the models are always in sync if the user does not edit the two different notations of the same model at the same time.

Challenge 2.3.5 is about how to achieve a metamodel for a textual DSL from the UML metamodel that has only the elements that are actually used by the company. This has been done by manual identification of metaclasses that are extended by the Ericsson DSL and metaclasses that are in use without being extended by any stereotypes. This worked because Ericsson's DSL rarely changes. However for companies where their DSLs change frequently automation of this step is a more suitable solution. There are approaches being researched on how to obtain this subset of UML automatically. For instance [55] suggested a way to split the UML metamodel into sub-metamodels according to the diagram type. But this still requires the users to identify key elements that are used in each diagram type. It also requires a user to build a tool that can do the splitting.

In [61], the authors propose a model shrinking approach that preserves the model elements' types. In this approach, an instance model that contains all the elements of the desired metamodel subset is needed. From this model, a selector tool is written that can extract the classes from the metamodel and create a metamodel subset. The metamodel subset is then shrunk to only the necessary elements needed. This method is usable if one has existing instance models that use all the classes that are needed in the subset of the UML.

Challenge 2.3.6 is about the fact that textual editors use qualified names to make references between elements. In our approach this has been solved by having a validation in the transformation that checks if model elements have names. If not we generate unique names for these elements. The names that we generate follow a pattern that corresponds to the type of model element so that a user reading the text model is able to understand it. For example if merge nodes are missing names we generate the names m1, m2, m3 and so on.

Challenge 2.3.7 is about how one can switch between models that have errors. In our approach this has been addressed by putting a layer of validation in the tool chain. Before transforming a model, one should validate the model first. If the model is inconsistent, then all errors have to be fixed before the model is switched. This is a prevention measure, thus future work is still needed on this.

2.6.2 Proposed Solutions for Non-Addressed Challenges

Our approach does not solve three challenges, which are challenge 2.3.1, 2.3.3 and 2.3.4 . However, during the study, developed a proposal on how these challenges can be addressed.

Challenge 2.3.1 which is about storage and versioning of models can be addressed in the following manner. First by the organisation putting in place a policy on which version of the model should be stored. For example they can decide to store the text or graphical version or both. Either way, all the designers should be aware of this policy. Second, with the policy in place then Version control software to support versioning of the stored models should

be put in place. Lastly if only one version is stored, automation mechanisms should be implemented to make sure those working with the other version that is not stored can work as smoothly as possible.

Storing both versions of the models also automatically solves challenge 2.3.4, which is model references since all the references in graphical models will be to graphical model elements and the links in textual models will be to textual model elements.

We propose addressing challenge 2.3.3 which is about graphical layout of the model and pretty printing using incremental model transformations. Incremental model transformations will work if both versions of the models (graphical and textual) are stored in the repository. Since the graphical and textual models are related by transformations, making these transformations incremental means that every time a transformation is executed, the target model will only be updated on parts that have changed. This will keep the layout of existing model elements in place.

Since we did not implement this solution, we cannot give definite answers about its effectiveness. Moreover, we are aware of the limitations that incremental transformation has, for instance when adding new model elements. These elements do not have any layout information and they could be placed anywhere in the diagram. The developer will have to arrange them manually when the transformation is run. This problem can be addressed by using automatic layouts which can be applied to the new and deleted model elements and preserve the layout of the already existing model elements as much as possible. This kind of auto-layout expands the model to make room for new elements and shrinks it when elements are deleted from the model [62].

Also incremental transformations sometimes fail to maintain consistency between models due to reasons like erroneous models, or changes in the source model that have more than one way of being propagated in the target model. This problem has been discussed in detail in [63].

2.7 Related Work

Due to the nature of the problem we explored, our related work is divided into two parts. The first part discusses related work on using both graphical and textual editors for a UML profile-based DSL while the second part is on how to integrate/combine UML profiles with Ecore-based tools.

2.7.1 Graphical and Textual Editing for UML

There are various approaches that have been investigated for editing UML models using textual syntax. Tools such as PlantUML [45], TextUML [64] and PlantText [65] all aim at establishing a standard textual language for UML as a GPL but none of them mention the use of profiles.

Other prototypes have been implemented to represent parts of the UML metamodel as text. For instance in [66], a textual editor for the Action Language for Foundational UML (Alf) has been developed based on Xtext. The textual editor is implemented in such a way that it can be used to edit only parts of a UML model and not the whole UML model. A different approach is proposed in [67], where textual editors are embedded in graphical editors. This way

when modelling using graphics, designers have an option to bring up a text editor as a pop-up box that they can use to edit model elements of graphical models. Our approach differs from these, as we want to be able to view and edit the whole model as a graphical or in textual model.

In [68] the authors describe an approach to use a textual editor to correct UML models that have errors. They implement a prototype using tUML, which is a textual concrete syntax for a specific UML subset. The subset supports class diagrams, state charts and composite structure diagrams. In this case the UML models are transformed to text and constraints are implemented in the textual editor. The developer fixes the errors in the textual editor and transforms the model back to graphical UML models. In this approach the transformations are not generated but written manually.

When it comes to switching between graphical and textual views, [69] proposes two approaches to facilitate the transformation of models that have both graphical and textual notation (i.e. one model containing parts written in UML and parts in text). The first approach is called Grammarware and refers to a text to text transformation of the models. With this approach the models are exported as text and transformation is done from text conforming to one metamodel to text conforming to another metamodel. The second approach is called Modelware and refers to a model to model transformation. In this approach, a model containing the graphical and textual content is transformed to a fully graphical model. This is done by converting the text part to its corresponding model element in the graphical metamodel. Our approach however proposes a way to have a DSL supporting both graphical and textual views and the possibility to switch between them but not combining text and graphics in the same model.

Projectional editing is another research area which investigates the use of various concrete notations for editing models. Projectional editing is a technology that displays the concrete syntax of models as projections. The concrete syntax of the model is not stored but only the abstract syntax of the model is persistent [70]. With projectional editing, the user edits the AST of the model directly and what the user sees on the screen is merely a projection. The main advantage of this technology is that the model can be projected in various notations depending on what the user prefers (textual, graphical, tabular or even a combination of these). However, this technology was not adopted in this study because it adds a lot of overhead to developers when it comes to learnability and familiarization [71]. Since projectional editing does not rely on parsers and hence has no grammar, it provides a different way of editing for which designers who are used to grammar based editors need time to get used to. Also, current reasonably mature projectional editors such as Jetbrains MPS [72] are also not yet integrated into Eclipse. For these reasons our approach uses Xtext which allows the use of grammar and provides functionality such as code completion and syntax highlighting out-of-the-box.

2.7.2 Bridging UML Profiles and Ecore DSLs

Due to the fact that one of the problems we had was how to combine UML profiles and Ecore, we also include related work on how to bridge UML and Ecore. The state of the art in this area is described in the following.

Most research on the field of bridging UML and Ecore DSLs have focused on the automatic generation of UML profiles from an Ecore-based DSL metamodel. Some notable examples in this context is research done in [73], [74], [75]. In these papers the main idea is how to systematically derive a UML profile from an existing DSL metamodel.

There also exists research on how UML profiles and Ecore DSLs can be used together. In [76] the author proposes a way to bridge UML and Ecore using model to model transformations. They use ATL HOTs and Atlas Model Weaver (AMW) for the transformation of models from UML profiles to Ecore and vice versa. This approach is quite similar to our approach but they do not generate the Ecore metamodel but assume that it already exists. In [77] another similar approach has been presented to bridge UML profiles and Ecore DSLs. In this work, the authors use a dedicated bridging language which is based on the AMU metamodel. The bridging language defines a model with mappings (weaving model) from UML profiles to an Ecore DSL and transformations are generated from this weaving model. Our work is different from this because we do not generate the UML profile but start with an existing profile. We also do not use the AMW model but a trace model whenever the mappings from UML to the Ecore DSL are not one-to-one mappings.

In [78], the authors describe how to interchange DSL and UML models using UML profiles. They propose an approach of first automatically generating a UML profile from a DSL metamodel (which can be in Ecore) using an integration metamodel. The integration metamodel is used to create models that define the relationship between the DSL metamodel and UML. From such an integration model, a UML profile representing the DSL metamodel is generated. During the generation of the UML profile, mappings from the integration model to the profile are also generated. The transformation from UML models to models conforming to the integration metamodel is done using the mappings generated when generating the profile. The transformation from the intermediate model to a model conforming to the Ecore DSL metamodel is done using the mappings from the Ecore DSL to the intermediate model which were manually created. This approach has the advantage of generating mappings from UML to the intermediate metamodel automatically but since the profile is generated from the DSL, it also means that the users using UML can only use classes that are extended by the profile. Any un-extended class will lack its mapping back to the Ecore DSL metamodel. Our approach solves this by including a subset of the UML metaclasses that are used even when the metaclasses are not extended by any stereotype.

In conclusion, most of the research on integrating textual and graphical editors for UML profile-based DSL is still on prototype level. Even though there are several text based tools implemented for UML as a GPL, the adaptation of these tools has not been reported in an industrial context, making it hard to know to what extent they can be used and what challenges they bring. Furthermore, the work reported on bridging UML and Ecore is also on a theoretical level accompanied with toy examples. None of the approaches report an industrial application of the bridge except the work of Jouault and Delatour in [68].

2.8 Conclusion and Future Work

In this paper we have presented an approach to integrate textual and graphical editors for a UML profile-based DSL using model transformations. We have shown that it is possible to have the two editors working in the same Eclipse environment. Even though this study has been carried out in one company, the results can be generalized to any company using a UML profile-based DSL that wants to additionally have a textual editor for it. We have also presented challenges that are encountered in industry when combining a UML profile-based graphical editor with a textual editor.

For future work we propose further research on how to effectively and automatically obtain a UML subset from the UML metamodel. Also research should be done to investigate ways to maintain the layout of graphics and format of text when using multiple editors. As mentioned previously this information is lost once a designer switches from one view to another and then back. Another aspect that should be researched is how to store models and how to keep these models in sync when working with version control tools such as Git. As some designers can decide to model using text and others decide to model using graphics, there has to be a standard policy for storage of these models. Should they be stored as text or as graphics or should one store both versions of the model? The research here could also shed light on what are the challenges when storing textual models or graphical models or both.

In connection to storage of models, strategies should be developed for merging models written using the different notations. For instance if one designer is editing the model in text and the other one is editing the same model using the graphical format, how will they merge these changes.

Inter-model referencing is also another area that requires further research. Inter-model references here means that one model has references to one or more elements in other models. For instance a graphical model can have references to other graphical models. When it comes to switching between views this has to be considered. Investigations are required on whether only one model should be switched to textual syntax and keep its references to the graphical models, or whether the referenced models will need to be converted to text as well.

Our study also investigated an efficient way of updating the DSLs and textual editor using transformations. However we did not look into how the updated DSL will affect the already existing models and how these existing models can be migrated to conform to the new DSL. This is a very important aspect and is also crucial future work.

Chapter 3

Paper B

Persisting Software Traceability Challenges in the Automotive Domain

S. Maro, M. Staron, J.-P. Steghöfer

In submission to Journal of Systems and Software

Abstract

In the automotive domain, the development of all safety-critical systems has to comply to safety standards such as ISO 26262. Such safety standards require traceability to be established between artifacts to ensure that resulting systems are well tested and therefore safe. Traceability which is the ability to relate artifacts created during development of a system, is therefore not only needed to keep track of the large number of artifacts produced but also required for safety certification. Despite the large body of knowledge on traceability, in practice establishing traceability in the automotive domain is still challenging. The aim of this paper is to understand what challenges still persist in practice, the solutions used, and how these relate to the published state of the art. To achieve this, we conducted a case study with a large automotive supplier and a tertiary literature study on the challenges of traceability. We found 19 challenges from the literature of which 16 were also found in our case company. Five of the challenges have been solved with solutions proposed in literature, five are partially solved, while six remain unsolved. We focus our discussion on unsolved challenges and propose solutions which could be feasible in practice.

3.1 Introduction

Over the past 20 years, the automotive domain has witnessed a tremendous increase of software deployed in cars. In today's modern car, software constitutes up to 40% of the production cost [79]. With upcoming trends such as autonomous driving, the software is not only getting more complex but also controls more and more safety-critical functions. The type of software has also shifted from small isolated functions to systems that contain several functions that interact and depend on each other [80]. Such complex systems can cause life threatening accidents when not properly specified, designed, implemented and tested. The number of artifacts produced during development (e.g., requirements, design models, behaviour models, simulations and tests) is large and their creation is usually distributed over various teams, including teams from different companies due to the OEM-Supplier relationship. With regards to the size of the systems, a typical high-end car consists of features that amount to about 100 million lines of code. This is a very large number as software in other domains has much less lines of code. For example, the F-22 fighter jet has about two million lines of code and the Boeing 787 has around 14 million lines of code [81]. In addition, it is not only the number of lines of code which is high in this domain but also the number of other artifacts. For instance the specifications of the systems in a 2004 car had already reached 20000 pages at that time [2]. This can be overwhelming if there are no standardized methods established to keep track of these artifacts, their relationships and how they evolve.

In such situations, traceability, plays an important role. In this paper, we define traceability as the ability to relate artifacts created during the development of a system, thus following [6]. Traceability helps in understanding which artifacts are connected to each other and how to keep track of which features have already been specified, implemented and tested. Traceability plays an even bigger role for maintenance tasks by facilitating change impact analysis and improving understandability of the system for the developer who needs to make changes in the system [82, 83].

In order to realize the benefits of traceability, software development companies need to establish a traceability strategy that is aligned with their goals. Defining and implementing a traceability strategy is not a trivial task, since it requires a good understanding of the artifacts to be traced as well as the ability to define meaningful links and to make sure the created links are useful [11].

On the one hand, there exists a large body of knowledge on traceability; for instance between 1999 and 2012, 70 studies on traceability were published in just the Requirements Engineering (RE) conference [35]. On the other hand, in practice, traceability is either not established at all [84] or only established since standards demand it [85] even in large software development companies. Our study therefore investigated the following research question:

RQ 1: How are the solutions proposed in traceability literature relevant for solving the challenges found in practice in the automotive domain?

To be able to answer this research question, we divided it into three smaller research questions as follows:

RQ 1.1: What are the traceability challenges and solutions reported in

literature?

RQ 1.2: What are the traceability challenges and solutions in practice in the automotive domain?

RQ 1.3: Which of the traceability challenges in literature are also evident in practice in the automotive domain and how have they been solved?

To obtain data for our study, we conducted a case study at an automotive supplier company and reviewed 20 secondary publications on traceability. We found 19 challenges from the literature of which 16 were also found in our case company. Five of the challenges have been solved with solutions proposed in literature, five are partially solved while six remain unsolved even though there are proposed solutions in literature. This paper extends our work reported in [86] in which we discussed challenges related to creation, maintenance and exchange of traceability. This paper discusses additional traceability challenges related to preparation and planning for traceability and the use and measurements of traceability. We also discuss in detail persisting challenges and give proposals for solutions viable in the automotive domain.

The rest of the paper is structured as follows: Section 2 describes the characteristics of software development in the automotive domain and Section 3 describes our research method in detail. Section 4 presents the the challenges and describes them from the perspective of the literature and what was found at the case company. Section 5 provides a discussion of the results focusing on the unsolved challenges. Section 6 discusses previous similar work, limitations of the study are discussed in Section 7 and Section 8 concludes the paper and outlines future work.

3.2 Software Development in the Automotive Domain

As previously mentioned in Section 3.1, the amount of software in cars is increasing at a fast pace. Software development in this domain is thus becoming important due to the economic value it contributes. Compared to other embedded systems development domains, development on the automotive domain has the following challenging characteristics:

[a] Heterogeneous nature of systems developed, ranging from comfort systems such as infotainment systems to safety-critical systems such as braking systems;

[b] Inter-dependencies between various functions deployed. There is an increase in the amount of sub-systems that need to communicate to make a certain functionality work. Krüger and colleagues give an example of the central locking system that has to interact with over 13 subsystems such as individual door controls, speed monitors, light controls and security alarms [87]. When not properly managed, this high-interdependency can lead to unintentional feature interactions that in turn results to failure [88].

[c] Large number of variants which makes it hard to manage, especially during maintenance where one car can have different versions of different systems installed. It is difficult to determine which versions are compatible [88].

[d] Development of long life products with a short time to market, and real-time systems that have special constraints on safety, security and reliability. This means that although there is a need to deliver fast inorder to remain competitive in the market, there are also very strict safety standards (e.g., ISO 26262) that need to be followed to ensure only safe products are released.

[e] OEM-Supplier relationship. Although some components are developed inhouse by the OEM, most of the components are developed by suppliers [89]. Due to the fast time to marker constraint, the components are usually developed by various suppliers and all these need to be integrated at the OEM. The task of the OEM has therefore evolved from assembling hardware parts to also integrating software systems [90, 91]. Due to this setting, the whole development process becomes complex since there are many people involved and likely located in different geographical locations which makes communication difficult. In addition, the OEM gives blackbox requirements to the suppliers which means that in case of failure during integration it is difficult to find errors in these subsystems and also harder to modify them [79, 88]. One way to reduce the complexity is having more than one level of suppliers for the OEM, the first tier supplier (also known as system suppliers [92]) have direct contact with the OEM and get contracts to deliver larger systems. These first tier suppliers then outsource parts or some of the components to second tier suppliers. Second tier suppliers can also outsource parts or some of the components to third tier suppliers [93]. This requires standardized interfaces that need to be agreed upon with all parties involved in order to ease the integration tasks. For safety-critical components, it is also important for OEMs to make sure that the suppliers involved do follow the safety standards in place.

3.3 Research Method

The aim of our study is to answer the following research question:

RQ 1: How are the solutions proposed in traceability literature relevant for solving the challenges found in practice in the automotive domain?

To answer this research question, we needed to collect the challenges from literature and from the real world. Therefore we used two types of research methods: a case study with an automotive supplier and a tertiary literature review. The case study provided data on which challenges exist in practice and their solutions if any. We conducted the case study according to the guidelines proposed in [37]. The tertiary literature review provided data on the challenges and solutions in the literature. Before conducting these two studies, we defined the scope that is relevant to us and which both studies will cover. Our scope (depicted in Figure 3.1) indicates that we distinguish three different traceability categories(*Preparation and Planing*, *Establishment*

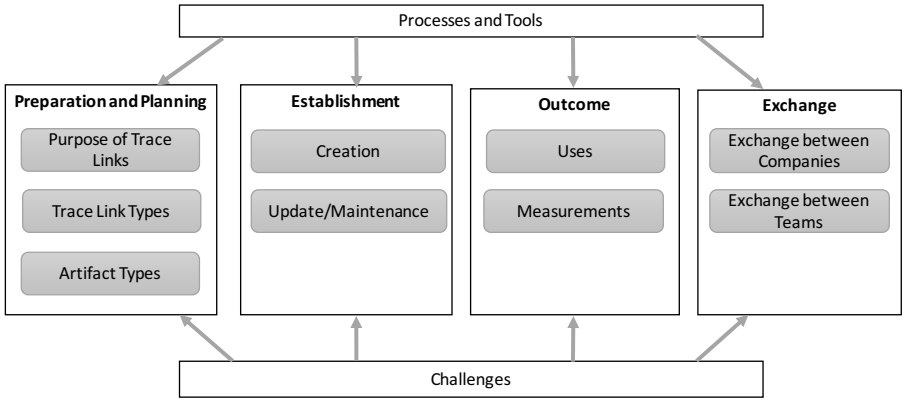

Figure 3.1: The scope of the study

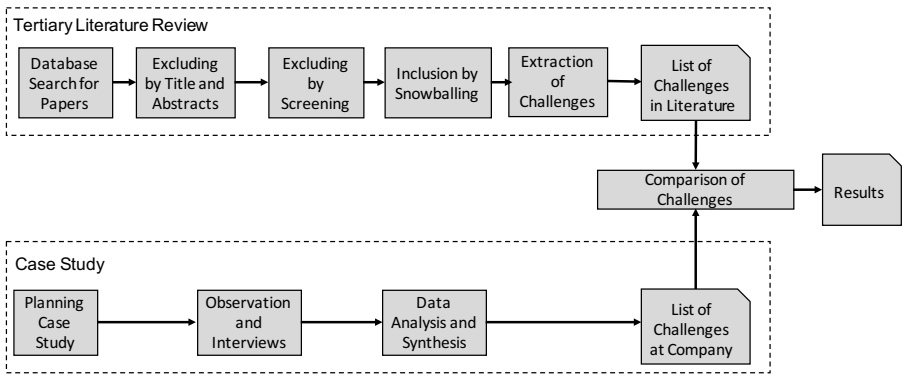

Figure 3.2: Summary of the Research Method

and *Outcome*) which are inspired by the generic traceability process model defined by Gotel et al. [11]. We used this model because it contains most of the activities needed for establishing traceability. This model is also well-known in the traceability community and since its definition it has been used in other research for instance in [26, 94, 95] as a basis for understanding and describing traceability.

In the model, the *Preparation and Planing* category, focuses on the processes and tools involved when preparing to include traceability in a company or a particular project. The *Establishment* category deals with the processes and tools involved in the actual creation and maintenance of traceability links. The *Outcome* category focused on how the links are stored and how they are actually used after they have been established. Since we are studying the automotive domain where the OEM-Supplier relationship means that artifacts are exchanged between companies, we added a fourth category called *Exchange* with the sub-categories exchange between teams and exchange between companies.

The details of the tertiary literature review are described in Section 3.3.1 and of the case study are described in Section 3.3.2. The entire research process is summarized in Figure 3.2.

3.3.1 Tertiary literature review

Our tertiary literature review followed the guidelines for conducting a systematic mapping study as proposed by [96]. The guidelines indicate that a systematic literature study should include five steps which are *Definition of research questions, Conduct search, screening of papers, Keywording using abstracts and Data extraction & mapping process.* The subsections below describes how these steps were carried out in our study.

3.3.1.1 Definition of Research Questions

Since the main aim of our research is to answer the this question *"How are the proposed solutions in traceability literature relevant for solving the challenges found in practice in the automotive domain?"*, from the literature study we had to identify both traceability challenges and solutions. Therefore our literature study has to answer the following sub research question:

> RQ 1.1: What traceability challenges and solutions are reported in literature?

3.3.1.2 Conducting the Search

Since this is a tertiary literature review, our aim was to find literature reviews published on traceability in the domain of computer science. We searched three databases : Scopus, ACM Digital Library and IEEE Xplore. The search string used was "Traceability AND (Literature Review OR Review OR Literature Survey OR Survey)" for Scopus and ACM Digital library and for IEEE Xplore the string used was "("Literature Review" OR "Review" OR "Survey" OR "Literature Survey") AND Traceability". This search led to a total of 222 papers which were reduced to 199 by removing duplicates.

3.3.1.3 Screening of Papers

By reading the title and abstract, we selected papers that are relevant to our study using the following inclusion criteria:.

[a] The paper reviews literature on traceability.

[b] The paper is published in a peer-reviewed venue.

[c] The paper is in the field of computer science.

[d] The paper mentions challenges of traceability and give a description of the challenges.

[e] The paper is in English.

The initial screening in which we read the title and abstracts left us with 21 relevant papers. After this we further read the introduction and conclusion of the papers and excluded eleven more papers because they did not fulfill criteria number one (The paper is reviewing literature on traceability) and four (The paper mentions and discusses challenges of traceability). From the remaining papers we also followed the citations (snowballing) to look for papers that

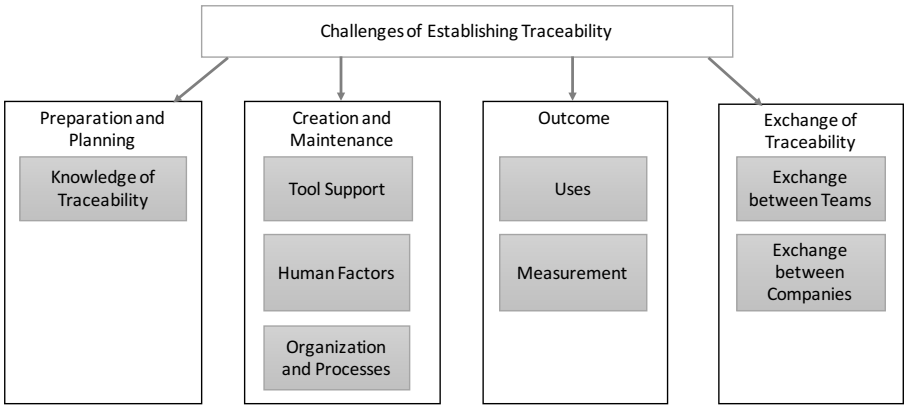

Figure 3.3: Categories of Challenges of Establishing Traceability

specifically researched challenges of traceability. This led to an addition of ten more papers. In the end, we had identified a set of 20 relevant papers.

3.3.1.4 Data Extraction and Classification

We examined all 20 papers, extracted the challenges and solutions they report and listed them in a spreadsheet. After this process, we reviewed all the challenges in the list and and devised the classification scheme depicted in Figure 3.3. In the *Preparation and Planning* category, all of the challenges found were related to the general understanding of traceability. For the *Establishment* category, the challenges related to *Creation* and *Maintenance* sub-categories overlapped significantly. We therefore merged the challenges of these sub-categories. Afterwards we reviewed the challenges and discovered that they could be further distinguished by technical issues in particular with the tool support, human factors that involved employee personality and others, and the organisational setting and established processes. The remaining two categories *Outcome* and *Exchange of Traceability* remained the same as in our conceptual model.

3.3.2 Case Study Design

This sub-section describes in detail how the case study was carried out. As previously mentioned, the study followed the guidelines on how to conduct case studies reported in [37]. The aim of the case study was to answer the following sub-research questions:

> RQ 1.2: What are the traceability challenges and solutions in practice in the automotive domain?

> RQ 1.3: Which of the traceability challenges in literature are also evident in practice in the automotive domain and how have they been solved?

3.3.2.1 Case and Subject Selection

The study was conducted in one of the world's largest suppliers of automotive components located in Germany. The company is multi-national which means that development is distributed in various locations. The company develops various types of automotive components ranging from hardware-only components to software-only components to embedded systems which include a software deployed on a certain hardware component. For this study we were interested in traceability during embedded systems development.

Our case study has two units of analysis within the same company. These are two departments both developing embedded systems at the company. We selected these two departments because they already implement traceability in their projects and develop safety-critical embedded systems for which traceability is a mandatory requirement. The two departments were also interested in improving their traceability practices, thus the topic was relevant and of interest to them. To be able to understand how traceability is implemented throughout the development life cycle, we conducted the study with seven participants in the following roles: two senior experts working on traceability (one from each department), four software system architects (two from each department) and one functional developer who belongs to one of the departments. We selected these roles in order to get a full picture on how development is done from when a requirement is received to when it is implemented and tested. The first role of senior expert is responsible for understanding what traceability needs the department has, surveying feasible solutions, acquiring these solutions and making sure that they are used in the department. The second role, system architect is responsible for receiving requirements from the customer, breaking them down and assigning them to development teams. This role is also responsible for managing the architecture of the systems that the department is developing. The last role, developer is responsible for implementing the features and testing them. In one department, the role of a developer and a tester are split into two separate roles assumed by separate people.

3.3.2.2 Data collection procedure

We collected data through observing demonstrations and conducting semi-structured interviews. Observations enabled us to understand the development process and how traceability activities are carried out and the semi-structured interviews enabled us to gather comparable data on the challenges. The model describing the scope of our study and interview questions were sent to the participants a week before the study took place. This was to allow them time to prepare for the demonstrations and interviews. For each participant, we started with the participant giving a demonstration on how they implement traceability using the scope model as a guide. This was followed by a semi-structured interview. The interviewer only asked questions which were not answered by the demonstration part. Due to legal issues, the interviews were not recorded but the interviewer took notes. The interviews and observation for each person lasted between 90 minutes to four hours with breaks in between. The longer sessions were with senior experts who explained and demonstrated the traceability process in detail. The interview guide for these interviews is

available online[1].

3.3.2.3 Analysis procedure

The data analysis started immediately after the observations and interviews were completed. This was to ensure that all relevant information was recorded for later analysis since the interviews were not recorded for legal reasons. The interviewer drafted a summary of the sessions and what was learned from the study and presented it to one of the senior experts for confirmation purposes. During this presentation, the interviewer described the development process and outlined the challenges that were learned from the interview. The senior expert could then confirm the findings or correct the findings when things were misinterpreted by the interviewer. The senior expert could also ask questions anytime during the presentation. This exercise led to few changes meaning that most of the initially collected information was correct. After this, we went through the interview notes and identified all the challenges. We used the categories in the interview model as analysis codes and placed each challenge found in the appropriate category.

3.4 Results

This section reports findings both from the tertiary literature review and the case study. The challenges found and their relationships are summarized in Figure 3.4. For each challenge, we first describe the challenge, discuss the solutions in literature and then compare them with the challenges and solutions at the company.

3.4.1 Preparation and Planning

This sub-section describes the challenges and solutions in that are encountered when companies are preparing to include traceability either in a specific project or the entire company. In this category, all the challenges identified are concerned with the availability and perception of the knowledge of traceability either by management of the company or the employees.

3.4.1.1 Knowledge of Traceability

We found four challenges related to knowledge of traceability from the literature and all these four were also found at the company. Two of the challenges have been solved by solutions that were also found in literature while two have only been partially solved using work-around solutions.

Lack of Knowledge and Understanding about Traceability

Description: In order to prepare and plan for traceability in a company, both the managers and developers need to have an understanding of what traceability

[1] `http://tinyurl.com/za392b6`

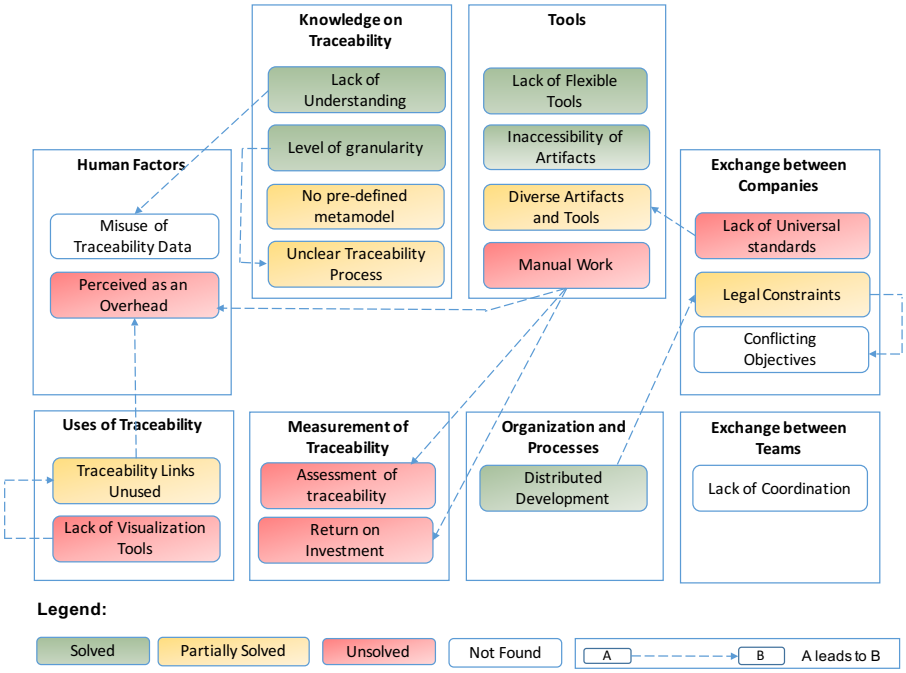

Figure 3.4: Summary of Traceability Challenges. The solved challenges have a green background, the partially solved challenges have a yellow background and the unsolved challenges have a red background. The challenges that have no background color were only in the literature and not identified in the case study. This means that the data collected was not sufficient to say if these challenges exist in the company. The directed arrows mean that one challenge leads to the presence of another challenge.

is and its purpose. This understanding also needs to be aligned, meaning that all the people in the company should have a common interpretation of what traceability is. For companies, if the concept of traceability is not clear, then the chances of failure are high.

Challenge and its Solutions in Literature: This challenge has been reported by seven papers from our review [9, 26, 35, 85, 97–99]. In [26], for instance, the authors report that some companies, especially those not working in safety-critical domain, have no notion of the term traceability. Another issue is that different individuals in the company have a different understanding of what traceability is for [9]. The most common is that managers see it as a mandatory task that needs to be done for certification purposes while developers perceive it as simply bureaucracy and a waste of time [85, 99]. In some cases where traceability tools are well established, developers may perceive it as important and useful for tasks such as impact analysis [98]. The literature proposes that in order to achieve a common understanding of traceability among all stakeholders, training is important. Early on, the company should invest some time and effort to train its employees on what traceability is and why they should do it. The training should also discuss what traceability

links are, what is complete traceability, what is traceability link quality and so on [99].

Comparison to Case Company: This challenge exists at the company but has already been solved. Given that the company operates in a safety-critical domain, employees are already aware of the concept of traceability. They base their understanding of traceability on the requirements defined by the safety standard they need to comply to (A-SPICE). They even have expert roles whose job is to understand what the standards require, form a strategy on what they need to do to comply, and communicate this to the rest of the company.

No Pre-defined Metamodel for Traceability in the Company

Description: Traceability links can be of different types depending on their purpose and which artifacts they connect. The link types can differ from domain to domain. Traceability link types are usually defined in what is known as a traceability information model or a traceability metamodel. Link types can be generic and carry little or not semantics at all (for instance a link type called "related_to" that allows connecting arbitrary artifacts) or they can also be specific and carry meaningful semantics (for instance a link type named "tested_by" that can only connect a requirement and a test in a sense that the requirement is tested by the connected test). Defining traceability links that carry domain specific semantics is advantageous as it allows for analysis of the links based on the semantics. In order to define the traceability information model, one needs to understand which types are needed and useful in the specific domain.

Challenge and its Solutions in Literature: This challenge was reported by three of the reviewed papers [9,25,100]. One of the solutions proposed is to define a standard traceability information model which has been done by [8] after making an observation in various companies. This model can indeed be used as a starting point for companies to define their traceability metamodel. However, since this is a domain-specific problem, another solution proposed is to document domain-specific guidelines on how to define metamodels [9]. This can be done through reporting case studies or experience reports.

Comparison to Case Company: At the company, this challenge exists and its is partially solved. In both the two departments, the traceability metamodel has already been defined. However, this has been done following the A-SPICE standard. This is a good start for the company as there is no other definition that they could currently follow. The downside is that the links are designed to fulfill the standard but there is no guarantee that this is sufficient to actually be useful to developers and architects developing the system. The standard also only suggests generic links that do not take into account the type of development the company has. For instance for systems developed as product lines, there is no information on what links should be added. The two architects interviewed reported that the company has product lines with a lot of variants but they do not know how to include traceability links that take into account variability. Another issue that is not addressed in the standard is

how to trace to non-functional requirements such as performance and security. The plan in the company is to use the current metamodel and collect data from its users on what is missing or which links are not working in order to customize the model for the company.

Level of Granularity

Description: When designing and planning for a traceability solution in a company or even a project, one question that arises is "what level of granularity should the traceability links created on?" For instance, should you link a requirement to a test file, a test case or a particular line of code in the test case? This is a challenge as if the links are too coarse grained then they do not give enough information and if they are too fine grained their number can become overwhelming and confusing to the end users.

Challenge and its Solutions in Literature: This challenge was reported in two studies [100, 101]. The solution suggested is that the granularity of the links should be defined explicitly in the traceability metamodel and the traceability links should be checked regularly to ensure that the links are created with the right level of granularity. This solution however does not suggest which level of granularity a project or company should use.

Comparison to Case Company: At the company, this was observed as a solved challenge. The company adopted the level of granularity suggested by the V-Model which is also suggested by the A-SPICE standard. As a solution, the system requirements are derived from customer requirements. The system requirements are then broken down into functional requirements which could be software requirements or hardware requirements. Software requirements are further refined into detailed software requirements. The developer is then assigned a detailed software requirement for implementation. Traceability links are created from customer requirements to software requirements to detailed software requirements. The detailed software requirement is then linked to an implementation file that actually contains the code. The detailed software requirement is also linked to a test.

Unclear Traceability Process

Desciption: Establishing a traceability strategy requires a traceability process (how links are created and maintained) to be put in place. Such a traceability process should be aligned with the software development process that already exists in the company. It is important for the traceability process to refer to work products of the existing development process. For instance, if a company defines requirements as user stories, then the traceability to and from requirements should refer to user stories and not something else that is not created in the development process. If such a process does not exist or is vaguely defined, links will be created in an ad hoc manner which results in low link quality.

Challenge and its Solutions in Literature: Three of the papers reviewed report this as a challenge [25, 98, 101]. In [101], the authors propose that the solution is to create a traceability process based on the traceability

metamodel defined at the company. This process should be documented and communicated to all stakeholders early on. Managers should be assigned the role of making sure that this process is followed. In [25], the authors propose putting in place an automated process of creating traceability links by generating skeletons of artifacts from requirements and their traceability links and let these skeletons be filled as development goes on.

Comparison to Case Company: At the company, this challenge exists and has been partially solved. A traceability process already exists and although it is a completely manual process, the developers and architects are aware of which links need to be created based on the breakdown of the requirements as discussed previously. In one department the requirements are defined as use cases and therefore traceability links are created from use cases to design, implementation and tests. In the other department the requirements are defined as user stories and therefore the links are created from low-level user stories to design, implementation and test. This challenge is partially solved as there are currently no roles that can check if the process for creating traceability links was followed. Sometimes during review meetings flaws of traceability links can be detected and fixed.

Table 3.1: Challenges associated with Preparation and Planning for Traceability

Challenge from Literature	Papers	Found at Company	Challenge Solved?	Solutions Match?
Lack of knowledge and understanding about traceability	[6, 9, 26, 35, 85, 98, 99]	Yes	Yes	Yes
No pre-defined metamodel for traceability in the company	[9,25,100,101]	Yes	Partially	No
Level of granularity	[100, 101]	Yes	Yes	Yes
Unclear traceability process	[25, 98, 101]	Yes	Partially	Yes

3.4.2 Creation and Maintenance

This section reports on challenges that are associated with the activities of creating and updating the traceability links. The challenges are divided into three categories which are tool support, human factors and organization & processes.

3.4.2.1 Tool Support

We found five major challenges in the literature which were reported in this category. Four of these challenges were also found at the case company. On further analysis (as shown in Table 3.2) only two of these challenges have been solved, one has a workaround solution, while two of them still remain unsolved.

Diversity of Artifacts and Tools

Description: In the software development life cycle there are a number of activities such as requirements engineering, system design and so on. In many

cases each of these activities utilizes a different tool and produces artifacts in different formats. Most traceability tools either do not support linking to artifacts located outside the tool or only support linking to specific tools and a specific format [6, 99].

Challenge and its Solutions in Literature: Eight of our reviewed studies report this challenge [6, 9, 82, 85, 98, 99, 102]. From the studies, there are two different solutions for this challenge. The first option is to use one tool that supports all the development activities. The advantage of such a *holistic* tool is that since all the artifacts are stored in one database they can be accessed for traceability link creation. The second solution is to integrate all the existing tools so that it is possible to create traceability links between them. This is however not a trivial task and requires a considerable effort especially if there are many tools that need to be integrated [20].

Comparison to Case Company: In the case company, there are a total of eight tools that are used for the different development activities. Tool integration is a technically challenging task. Therefore, the company currently uses implicit links to link to artifacts in different tools which are created by copying IDs from one tool to another. This is not only time consuming, but also error prone and does not allow for any analysis to be done on the links. To overcome this problem, the company is planning to acquire a holistic tool that will be able to store all of their artifacts and thus make them accessible for creating traceability links. The main drawback of this solution as reported by one of the architects is that it is hard to find a holistic tool that fully supports all the activities the development life cycle. Currently, there are no holistic tools supporting activities like simulations which means that even with the holistic tool in place, other tools will still be used. Therefore this challenge is partially solved as linking to tools outside the holistic tool requires implementation of special plugins, which is costly in terms of time and might require rework as the involved tools evolve.

Manual Link Creation and Maintenance

Description: The task of creating traceability links is one that is time consuming especially when it is done manually. Moreover, traceability links immediately become outdated when the artifacts they connect evolve. This means that they also need to be updated in order to remain correct. Updating them is also time consuming and most of the time error prone.

Challenge and its Solutions in Literature: This is one of the most frequently reported challenges in the literature. In our review it has been reported by ten out of 15 papers [6, 9, 10, 19, 26, 35, 82, 85, 99, 100]. To overcome this challenge, the literature proposes the use of automated techniques to generate and update the traceability links. Examples of these techniques are machine learning [103], information retrieval [104], event-based techniques [99] or model-driven techniques [105]. Most of the studies reporting these approaches have been on a theoretical level with small examples and using students as test subjects. For instance the literature review conducted by Borg et al. on information retrieval approaches for recovering traceability links show that out of 34 publications studied, only one had an industrial evaluation [104].

Comparison to Case Company: Interestingly, none of these solutions was viable for the company. Generally machine learning, information retrieval and event-based techniques have a low precision and therefore the chance that false traceability links are generated is high. Given that the company produces safety-critical systems and the traceability links are also used for the certification process, false links are not tolerable. Model-driven techniques, on the other hand, require that all the artifacts being linked to and from are represented as models which is not the case for the company, where only some of the artifacts are models.

Lack of Flexible Tools

Description: Since traceability link types can greatly differ from company to company or even project to project, it is crucial for the tools to allow for custom traceability link types to be defined. Providing a tool that can only be used in a specific context is a limiting factor. Tools need to allow for customization of which links can be created depending on the users' needs.

Challenge and its Solutions in Literature: This challenge was only reported by one study in our review [85]. The solution described is urging developers of traceability tools to take into account how flexible the tool should be. For instance traceability tools should be flexible in a sense that they allow definition of custom links, allow linking to arbitrary artifacts, be able to define which reports should be created from the links and so on. The more flexible the tool, the better as companies can tailor it to fit their project needs.

Comparison to Case Company: This is one of the challenges that the company has solved. For requirements management, they have adopted DOORS[2], a tool that is flexible and allows for definition of custom traceability links. Out of the box, the tool allows definition to different types of links to link to and from requirements. Linking to other artifacts that are stored outside the tool can be done through OSLC[3] (Open Services for Lifeycle Collaboration) which is a standard for sharing artifacts across tools. For artifacts that do not have OSLC representations, special attributes in the requirements can be defined to store IDs or names of artifacts that are outside the tool. While OSLC enables creating links to artifacts in external tools, maintaining consistency of these external links is a challenge as when artifacts evolve in their tools, these changes are not propagated to DOORS for the links to be updated accordingly.

Inaccessibility of Artifacts

Description: When creating or updating a traceability link, it is crucial to have access to the artifacts that need to be connected by the traceability link. In a situation where a project contains a large number of artifacts, tool support is needed to assist in locating the different artifacts. It is very cumbersome if one has to search through hundreds or even thousands of elements manually.

Challenge and its Solutions in Literature: Only one of the reviewed papers mentioned this challenge [15]. The solutions proposed is that the com-

[2] `http://www.ibm.com/software/products/en/ratidoor`
[3] `http://open-services.net`

pany, through tools, should ensure that users have all the necessary information and access to the artifacts needed to create traceability links. Tools should provide features such as search by ID or keywords, to make it easy for the users to find the artifacts they need.

Comparison to Case Company: For the case company, this is not a challenge as the tools they use have the ability to search for and locate specific artifacts in an easy way. For traceability links involving artifacts stored in different tools the user still needs to copy the ID from one tool to another. Moreover, every tool has a search functionality.

Table 3.2: Challenges associated with tools

Challenge from Literature	Papers	Found at Company	Challenge Solved?	Solutions Match?
Diversity of Artifacts and Tools	[6, 9, 82, 85, 98, 99, 102]	Yes	Partially	Yes
Manual Link Creation and Maintenance	[6, 9, 10, 19, 26, 35, 82, 85, 99, 100]	Yes	No	
Lack of Flexible Tools	[85]	Yes	Yes	Yes
Inaccessibility of Artifacts	[15]	Yes	Yes	Yes

3.4.2.2 Human Factors

In this category we found two main challenges that have been reported in the studied literature. As shown in Table 3.3, only one of these challenges was found at the case company.

Misuse of Traceability Data

Description: This challenge refers to the fact that in some situations, people responsible for creating and maintaining the traceability links have a fear that this data may be used against them, e.g., during performance appraisals. This happens especially when developers need to create links from artifacts they are responsible for to for example bugs reported by users.

Challenge and its Solutions in Literature: This challenge has been reported by three of our reviewed literature [9, 85, 98]. The authors describe that employees have a fear that traceability data can be used against them and threaten their job security. This is an inappropriate use of traceability data as the data is supposed to be used for quality assurance of the system rather than used for judging employees' performance. The studies propose that both management and employees need to be educated on what traceability is and what the potential benefits are.

Comparison to Case Company: At the case company, this was not part of the challenges that we identified. However, the company has a system that already logs user activities with respect to creating and modifying development artifacts. If there is a problem in the system it is easy to identify who was working on the artifact and contact them about the problem. This data is

not used for performance appraisals. This indicates that the development environment is already very transparent thus employees do not have this fear of misuse of traceability links.

Perceived as an Overhead

Description: In situations where traceability links are created manually, developers usually perceive this as an extra activity that they need to do on top of their daily work. Since the people creating the links are often not the ones that end up using them, they see it as doing a job that only benefits other people. This is a problem as they become demotivated and assign a low priority to this task, which can lead to either wrong or missing links.

Challenge and its Solutions in Literature: Four of our reviewed studies report this challenge [15, 26, 98, 99]. Proposed solutions for this problem are to ensure that the traceability links created provide immediate benefit to the user who is creating the links. This can be done with tools that enable quick navigation from one artifact to another or visualization techniques that give users an overview of the connection between different artifacts.

Comparison to Case Company: At the case company this is a challenge, due to the break between tools and the fact that implicit links are created between artifacts in different tools. It is hard for developers to get an overview of the traces. Across tools they still have to find artifacts by searching for ID thus do not see the immediate benefits of traceability. All of the interviewees pointed out that being able to navigate easily using the traceability links and having graphical representations of how everything is connected would be a feature that would encourage them to create more correct and complete traceability links. Allowing for easy navigation across tools requires integrating the tools which is also not a trivial task as previously discussed.

Table 3.3: Challenges associated with Human Factors

Challenge from Literature	Papers	Found at Company	Challenge Solved?	Solutions Match?
Misuse of Traceability data	[9, 85, 98]	No		
Perceived as an overhead	[15,26,98,99]	Yes	No	

3.4.2.3 Organization and Processes

In this category, we found only one challenge and this challenge has been solved (see Table 3.4).

Complexity Added by Distributed Software Development

Description: In large organizations, it is a common phenomenon that development activities are carried out at multiple sites. This adds complexity to

traceability especially when the different sites need to share the development artifacts. If the development infrastructure is not well set up, it can be very hard to create traceability links between artifacts that are produced in different locations.

Challenge and its Solutions in Literature: This challenges has been reported by two of the reviewed papers. These papers propose a centralized repository for storage of all the development artifacts [9, 15]. This way the location of the developers will not matter as everything is centrally stored and shared. Such a repository also needs to be guarded by an access control system to make sure that the right people have access to the artifacts they need.

Comparison to Case Company: Essentially, this is not only a traceability problem, but a distributed software development problem in general. The company has solved this challenge by having centralized repositories where the artifacts can be stored and different developers are given access rights accordingly. This is in line with what the literature proposes.

Table 3.4: Challenges associated with Organization and Processes

Challenge from Literature	Papers	Found at Company	Challenge Solved?	Solutions Match?
Distributed software development	[9, 15]	Yes	Yes	Yes

3.4.3 Outcome

In this section, we report on challenges that are related to the outcome of the traceability process. The section is divided into two subsections which are *Use of Traceability* containing challenges encountered when using traceability links and *Measurement* containing challenges associated with measuring the quality of the traceability links.

3.4.3.1 Uses of Traceability

For this category, we found two challenges. One of this challenge has been partially solved and one challenges is unsolved.

Lack of Proper Visualization Tools

Description: When traceability is properly established, it can result in a large number of links, in particular if the project consists of a large number of artifacts. The end users of these links need proper visualization tools in order to understand them. This is currently a challenge as traceability links are usually presented in large tables or lists where it is hard to comprehend what they mean and even harder to detect flaws in them.

Challenge and its Solutions in Literature: This challenge was reported by three of the reviewed papers [35, 99, 106]. In [106], the authors

point out that especially with automatically generated traceability links, it is important to have meaningful graphical representations so that traceability links can be easily inspected for inconsistent and outdated links. Visualization techniques that will facilitate development activities are proposed in [35]. For instance, it is useful to have a visualization that will allow the user to see which requirements are already implemented and tested or which tests do not have corresponding requirements.

Most common visualizations of traceability links are the matrix, hyper links and graphical notations. In the matrix view artifacts are displayed in a table with a mark on the cell where the artifact in the column and that in the row are connected by a traceability link. In the hyperlinks view, traceability links are displayed as hyperlinks from an artifact and can be clicked to navigate to the connected artifacts. The graphical view represents the artifacts as nodes and the links as edges in a graph. The authors in [99], propose that a traceability tool should have a combination of the three representation as all have advantages and disadvantages and are used for different purposes. The authors illustrate that a project manager may need only an overview of the project but a developer making a change to the system may find hyperlinks more useful as navigation to and from artifacts is facilitated [99].

Comparison to Case Company: In the case company, this was also reported as a challenge that is not solved. This was mainly noted by the developer and the architects who suggested that the traceability links would be more useful for them if they had better graphical representation. They specifically asked for visualization where one is able to get an overview of the project or a specific feature through the traceability links. Also the traceability links that are created manually, for example by copying an ID of one artifact and adding it in another, are not supported by the visualization available in the requirements management tool used in the company.

Traceability Links are Almost Never Used

Description: It has already been discussed that establishing traceability links takes a lot of effort and time. However, even with the amount of time invested, apart from certification purposes, traceability links are either not used at all or under-utilized. This is mainly due to lack of tools that facilitate utilization (for instance, good visualizations) and lack of trust in the quality of the traceability links maintained.

Challenge and its Solutions in Literature: This challenge has been reported by two of the reviewed papers [99, 101]. In [99], it is reported that traceability links are not used either because the links recorded are not helpful to support development activities or because the tools do not provide an efficient way of using the links. The authors point out the importance of tailoring traceability according to the needs of the users and not just creating traceability links for every artifact. In [101], the authors point out common flaws that cause traceability links to be ignored. These flaws are for instance, redundant traceability paths (mulltiple ways to define tracebility links from one artifact type to another) which may be inconsistent, missing links, out-dated links and traceability links being presented in large tables that are hard to

comprehend.

Comparison to the Case Company: At the company, the main drive for establishing traceability is due to the requirement from OEMs who need to be A-SPICE compatible. Therefore the main use of the traceability links is for certification purposes. During the interviews we also found that traceability links are used to mainly track the progress of the project, for instance, to check how many requirements already have test cases. The architects and developers however noted that they would like to utilize the links more but that there is no convenient way to do that at the moment. For instance, it is sometimes necessary to copy IDs from one tool to another to search for the connected artifact. This makes it very hard to get an overview of the system or feature through the traceability links. This challenge is therefore partially solved and would be fully solved if better tools that facilitate usage of traceability links are put in place.

Table 3.5: Challenges associated with Uses of Traceability

Challenge from Literature	Papers	Found at Company	Challenge Solved?	Solutions Match?
Lack of proper visualization tools	[35, 99, 106]	Yes	No	
Trace links are almost never consulted or used	[99]	Yes	Partially	Yes

3.4.3.2 Measurement

For this category, we found two challenges and all of them are unsolved.

Difficult to Assess of the Quality of Traceability Links

Description: As previously mentioned, when traceability is properly established, it can result in a large number of links. In order to trust and use the traceability links, it needs to be possible to assess their quality by for instance measuring how correct and complete the set of traceability links is. This is a challenge as the most reliable assessment method is still manual checking.

Challenge and its Solutions in Literature: Three of our reviewed papers note this as a challenge [6, 35, 101]. It is hard to assess if the traceability maintained is of high quality as reported in [101], where the authors note that even in safety-critical domains the traceability links submitted for certification contain either missing links or redundant links. In [6], it is reported that especially for generated traceability links, it is a challenge to evaluate their correctness and completeness. One proposed solution is to attach confidence values to the generated link and have a threshold based on the confidence value to determine which links are correct. However, this approach does not guarantee that the links will be complete or correct. Another solution

is to use the semantics defined in the traceability metamodel to assess the traceability links. For instance if the metamodel defines that every requirement should be linked to a test, then completeness can be assessed by checking if all requirements have a link to a test. This however only guarantees completeness and correctness will still need to be checked manually.

Comparison at the Case Company: At the case company, this is currently one of the unsolved challenges. For traceability links that are created between artifacts in DOORS, there is a possibility to check for completeness easily since the tool allows identifying requirements with no links. Also since the tool supports defining custom trace links, it is possible to limit which kinds of artifacts a link can connect. The advantage of this feature is that it prevents the creation of links that are semantically wrong. For links that are created with artifacts that are not in DOORS this kind of check is harder as it requires implementation of extra plugins that can do such checks. Correctness on the other hand is still a problem and needs to be checked manually. This can be done during review meetings but consumes a lot of time and effort.

Difficult to Measure the Return on investment

Description: Since the most common way of establishing and maintaining traceability in practice is manually, this is a cost-intensive task that requires the company's investment both in terms of money for the tools and in terms of time. It is therefore important for a company to be able to measure what is the return-on-investment of the traceability links established. This is a challenge as the cost is significant while the benefits of it cannot be easily measured.

Challenge and its Solutions in Literature: Seven out of the reviwed papers report that traceability establishment is an expensive process [9, 10, 15, 82, 85, 99, 100]. This is because developers need to spend extra time to create and maintain traceability links. Most managers think that a project that implements traceability is more expensive than one which does not [85]. Currently there are no measurements that can provide evidence of these direct benefits of traceability. Research proposes cost-benefit models that can be used to show how much traceability has contributed to activities such as maintenance and understandability [82], but these still need to be validated in practice. This is not a trivial task as such benefits are mostly visible at the end of the project. To minimize the effort spent on traceability creation and maintenance, researchers have proposed Value-Based Traceability, which means tracing to only high priority requirements as compared to full traceability [85].

Comparison to Case Company: The results of the case study indicated that this challenge has not been solved. All of the interviewees including the managers confirmed that they think traceability is expensive and they do not have evidence of the value it adds to the projects. The only reason that justifies investing in traceability is because it is a mandated task, they have to do it. Value-Based Traceability is also not a feasible solution for them as *full* traceability is a mandatory requirement for safety-critical applications. It is also hard to maintain an exclusive list of high priority requirements that need traceability as priorities can rapidly change over time.

Table 3.6: Challenges associated with Measurements

Challenge from Literature	Papers	Found at Company	Challenge Solved?	Solutions Match?
Assessing the traceability maintained	[6, 35, 101]	Yes	No	
Return on Investment (ROI).	[9, 10, 15, 82, 85, 99, 100]	Yes	No	

3.4.4 Exchange of Traceability Information

In this category we found three challenges from the surveyed literature. Two of these challenges were also found at the company where one is partially solved and one is unsolved as shown in Table 3.7.

3.4.4.1 Exchange between Teams

Lack of Coordination in traceability activities

Description: During software development different people with different roles need to coordinate in order to work together. This becomes more important in system development because various parts of the system are developed by different teams and have to be integrated in the end. For example the software team needs to coordinate with the hardware team to make sure that their software will work on the hardware. This coordination is also important when it comes to updating the traceability links. Different teams working on artifacts connected by traceability links need to coordinate when maintaining the links.

Challenge and its Solutions in Literature: This challenge was observed by two of the papers we reviewed [10,95]. In [10], value-based traceability is proposed as a means to reduce the amount of links created and hence reduce the time people need to coordinate on traceability link maintenance. In [95], the authors report that change notification is a very useful feature for coordination. When an artifact connected by a traceability link has been changed, then the person responsible for the connected artifact should get a notification of the changed artifact in order to make a decision on how the link should evolve.

Comparison to Case Company: At the company this was not observed as a challenge. On further analysis this can be due to the fact that the requirements management tools has a feature called “suspect links”. It highlights the links that connect artifacts which have changed. The user can thus investigate the change and decide how to update the traceability link and the connected artifacts. When working as a team, the suspect links are also propagated to a developers local workspace when they pull changes from the repository. The developers can navigate to see what has changed in connected artifacts by clicking the suspect links.

Table 3.7: Challenges associated with with Exchange of Traceability between Teams

Challenge from Literature	Papers	Found at Company	Challenge Solved?	Solutions Match?
Lack of Coordination in traceability activities	[10, 95]	No		

3.4.4.2 Exchange between Companies

In this category, we found three challenges, two of which were also identified at the company. One of the challenge has been partially solved even though there was no proposed solution in literature and one is still unsolved.

Legal Constraints

Description: As mentioned before, in the automotive industry, development activities are distributed between the OEM and different suppliers. This implies that the different artifacts produced are also distributed. Establishing traceability links that cross the organizational boundaries is a challenging task due to legal and privacy implications. Some artifacts can be inaccessible to the supplier because they are confidential to the OEM.

Challenge and its Solutions in Literature: In the reviewed literature, two of the papers [95, 107] mention this challenge but there are no proposals for how to establish traceability when the artifacts are restricted due to legal reasons.

Comparison to Case Company: The company also faces this challenge when some of the artifacts they want to trace to cannot be shared by the OEMs. Currently they do not have a solution for this. For some OEMs, the company shares requirements (for instance in XML format) via web interfaces. The OEMs can then limit which has fields can only be visible to the OEM and fields that can be visible to both the supplier and OEM. This is an initiative towards a sharing of confidential information.

Lack of Universal Standards

Description: To facilitate the sharing and transfer of traceability information from one company to another, there is a need for a common standard. Currently this does not exist and traceability information exists in various forms ranging from implicit links established through copying IDs from one artifact to another, to explicit traceability links that utilize formal notations such as models. Some links are also stored together with the artifacts while others are stored in a separate trace model with only references to the connected artifacts. Depending on the tool the formats of the traceability links can also vary substantially.

Challenge and its Solutions in Literature: The literature proposes the need for one standard that can be used by companies in order to facilitate

this sharing and exchange of traceability information [82].

Comparison to Case Company: This is a challenge that the company faces. For instance, OEMs can send requirements which could have traceability links as well. But if the tools at the company cannot identify these links then that information is lost and has to be created from scratch.

Conflicting Objectives

Description: When more than one company is involved in the development of a system, it is important to align organizational objectives of all the companies. This is true also for traceability. If the objectives for traceability in one company contradict the ones in another, there might be a conflict. For example if the supplier requests traceability information that is conflicting with the OEMs objectives (for instance violates privacy policies), then this will not work.

Challenge and its Solutions in Literature: Only one of the reviewed papers [107] reports this challenge. In [107], it is proposed that at the beginning of the project, all the stakeholders need to align their objectives, including traceability objectives. It is important to define early on what each stakeholder requires and is expected to deliver in terms of traceability.

Comparison to Case Company: This challenge did not come up in the study at the company. Since the company is a supplier, one of their objectives is to satisfy the OEM. In this case, the demand for traceability actually comes from the OEMs. The OEMs specifically asks the company to be compliant to the A-SPICE standard in which traceability is one of the requirements.

Table 3.8: Challenges associated with Exchange of Traceability between Companies

Challenge from Literature	Papers	Found at Company	Challenge Solved?	Solutions Match?
Legal Constraints	[95, 107]	Yes	Partially	
Lack of universal standards	[82]	Yes	No	
Conflicting objectives	[107]	No		

3.5 Discussion

From the findings reported in Section 3.4, the challenges fall into three different categories: solved challenges, partially solved challenges, and unsolved challenges. An overview of this is given in Figure 3.4. Although the partially solved challenges are also interesting to analyze, in this discussion we focus on the unsolved challenges to understand why they are not solved and propose solutions that could be investigated to solve these challenges. Our analysis is based on the case studied and therefore limited to the automotive domain. Table 3.9 gives a summary of the persistent challenges and the solutions that we propose.

In the *Tools* category the unsolved challenge is *Manual work* of establishing and maintaining the traceability links. Several studies have focused on machine learning [103, 108], information retrieval [109] and rule-based techniques [110] for automating the creation and maintenance of traceability links. However, the chance that incorrect links are generated or links are missing is still high which is a hindering factor for applications of such techniques in a safety-critical domain, like the automotive domain. To overcome the problem of incorrect links, researchers have proposed a solution where the generated links are manually inspected by humans to remove links that are not correct. However, this is not guaranteed, as in [111] the authors show that giving a set of generated links to humans to sort out incorrect links, can lead to a worse set of traceability links. Therefore, due to the fact that automated techniques can generate incorrect links, which is in violation to safety standards such as ISO 26262, they have not been adopted in the automotive domain.

Other automation techniques in literature are model-based techniques for which traceability links are generated as a by-product of the transformation. The drawbacks of model-driven traceability is that first it assumes that all the artifacts are models, which is not the case in the automotive industry. Even if models exist, they are independent, not connected by transformations. Secondly, many transformation tools that support the generation of traceability links have their own pre-defined notion of what the links should be. This makes it hard to integrate them in existing traceability tools already used in companies [25].

To practically solve this challenge, traceability tools have to enable the combination of manual, semi-automatic and automatic techniques for creation and maintenance of tracebility links. This is because all these approaches have their advantages and disadvantages and can complement each other. For instance to make sure the links are correct, one can rely on manual creation, but to reduce the effort of maintenance automatic and semi-automatic techniques can be used. Semi-automatic techniques include sending notifications and warnings to users on traceability issues and suggesting probable solutions on how to fix issues. This kind of solution has been investigated in [22, 112] and the authors show that the solution is promising when properly integrated into the traceability tools. Moreover, to leverage model-driven approaches of link creation and maintenance, the tools need to be able to combine links created from transformations with the manually generated links so that they can all be used together.

In the Exchange of Traceability category, the unsolved challenge is that there is *no common standard for exchanging of traceability links*. To solve such a challenge, both practitioners and researchers need to work together to establish a traceability standard. For requirements, there is already a Requirements Interchange Format (ReqIF)[4], which is being adopted and provided as exports from several requirements management tools. Extending such a standard or creating a similar standard for traceability exchange will resolve this challenge. It should be noted that, having the standard in place will not solve the diversity of tools problem as data will still need to be exported from one company and shared with another which can cause inconsistency issues as the data evolves. Where not legally constrained we encourage suppliers and OEMs to share the

[4] `http://www.omg.org/spec/ReqIF/1.1/`

Table 3.9: Proposed solutions for the unslved challenges

Challenge	Solution in Literature	Proposed Extensions
Manual Work	Machine Learning [103], Information Retrieval [104], Rule-based [110] and Model-based techniques [105]	Combine manual links with model-based techniques to create links. Use semi-automatic approaches for maintenance (e.g., to push notifications of artifact changes to responsible users and suggest how links should be updated)
Perceived as an overhead	Develop tools that require less effort and produce immediate benefits (e.g, ease of navigation), training on importance of traceability [15, 26, 98, 99] .	Complement the traceability process with gamification features. For instance developers can be rewarded based on the number of correct links they create and projects can be awarded points/badges based on completeness of traceability links.
Lack of visualization tools	Matrix view, Graphical view and Hyperlinks [99]	Provide visualizations suitable for end users needs. This can be done by developing tools that enable visualizations to be customised. Users should be able to create different views (graphs, charts, matrices, etc.) based on different data from traceability links.
Assessment of traceability	Using a well-defined traceability metamodel that can facilitate completeness checks and prevent invalid link creation [101], event-based maintenance [112], text-matching	Extend event-based techniques to enable notifications to be sent to artifact owners when links are created involving these artifacts in order to facilitate correctness checks
Return on investment	Value-based traceability [113]	Monitoring activities supported by traceability to automatically collect evidence on advantages of traceability. Communicating this evidence in the company
Lack of Universal standards	Create a traceability standard [82]	

data repository to avoid such inconsistencies.

In the *Use of Traceability* category, the unsolved challenge is *Lack of Visualization Tools*. At the case company, all the interviewees were not satisfied with the visualization provided by their traceability tool. Our analysis shows that this is attributed to the fact that most tools are not well adapted to the requirements of using links in different scenarios. Instead, much of the effort in developing these tools is dedicated to the functionality of creation and maintenance of the links, rather than visualization. To solve this problem, we propose that there is a need to first analyze different use cases in which traceability links are used. An old study by Gotel et al. investigated different scenarios in which traceability links are used. Conducting such a study in the automotive domain will lead to usage scenarios that can be used to determine which kind of visualization is appropriate for each use case. When this is clear, it will be possible to add such visualizations to existing tools and support the users when using traceability links.

In the Measurement of Traceability category, both challenges identified are unsolved. The first challenge is *Assessment of Traceability* which refers to how the quality of the maintained traceability links can be measured to ensure that the links are both correct and complete. Measuring completeness is attainable as long as the definition of completeness is clear in the organization and the tools are able to include this definition. For instance a completeness metric of traceability can be "every requirement should be linked to a test case". If there is a traceability link defined that links requirements to tests, then the tools are able to filter out requirements with no links to test cases and flag these for the people responsible.

Correctness on the other hand is harder to assess with tools. For instance a requirement can indeed be linked to a test but in order to tell if the test is a correct test for the requirement, manual assertion needs to be performed. This is a very time consuming task. To tackle this problem, text-matching [114] is one of the solutions that can be used to reduce the time spent on this task. For instance a set of traceability links can be analyzed by a text-matching algorithm to get a similarity score between the connected artifact. The links with no similarity at all can be shown to the user for a manual check of their correctness. This solution will only work if there are naming standards in place and that ensure that there is always a text similarity between two connected artifacts. Another solution to check for correctness is to have notifications when links are created. This means that if a user creates a traceability link between A and B, both the owners of artifacts A and B are notified of this traceability link and can raise their concern if they think the link is incorrect and discuss the link with the user who created it. This approach is similar to event-based traceability proposed in [115] where the authors propose notifications to be sent to the owners of connected artifacts when one connected artifact evolves in order to update their artifacts too. We propose an extension of this event-based traceability approach to include notifications when the traceability links are created.

The second challenge in the Measurement of Traceability category is *Return on investment*. This refers to measuring the benefit that traceability brings to a project and the company in general. Most literature on traceability points out benefits such as saving time and effort during impact analysis, tracking

progress and improving understandability of the system. However, measuring these benefits in an industrial setting is not a trivial task because it is hard to isolate the effect of traceability. Also traceability benefits are visible once the project has been going on for a while as developers leave the project and those who remain forget things about the project. Here traceability is seen as beneficial as it saves time by helping developers to understand the system and easily navigate to artifacts. Value-based traceability is one solution proposed to reduce the cost of creating traceability links [113, 116]. This means that when planning for traceability, the companies need to assess why the links are needed and how they will benefit from them. This will lead to links that are actually useful for the project and thus beneficial.

In the automotive domain, the main reason for adopting traceability is due to safety standards that demand traceability. This is however not a good motivation as traceability is adopted because people are forced to do it and not because they want to do it. Being able to quantify the benefits of traceability is one way to show that traceability is indeed useful. For this we propose monitoring the activities that are supported by traceability links in the company in order to get data on how traceability links are useful. Additional data can be obtained by conducting surveys with users of traceablity and publicizing the results internally in order to promote its adoption in the company even for projects that are not safety-critical and thus controlled by safety standards.

In the Human Factors category, the unsolved challenge is that traceability is *perceived as an overhead*. This challenge has two aspects: an organizational and a technical one. The organisational issue is that the people creating and maintaining the traceability links are not the ones using them. A relation to the challenge of understanding traceability thus exists and sufficient training as well as the realization of the immediate benefits of traceability links can help in this regard. The technical aspect is related to the tools that are in use and that offer little support in terms of visualisation, navigation, and analysis. If, based on traceability links, the tools used in the industry can offer features such as easy navigation, visualization, customized reports or even recommendation for artifacts that can be re-used, then the developers creating the links will see their benefits. It should be possible to customize the tools in a way that benefits the creators of the links as well [117]. Another idea which we propose is complimenting traceability tools with aspects of gamification to make the task of creating and maintaining the traceability links more motivating and engaging. This has been shown to work with other software engineering tasks such as requirements analysis and testing [118].

3.6 Threats to Validity

In this section we discuss the threats to validity of our study and ways in which we minimized these threats. We use the categories described in [37] but do not discuss internal validity as our study was not not examining a causal relation.

3.6.1 External Validity

This threat refers to how generalizable the results of the study are. In our case study, we applied data triangulation and interviewed seven employees of three different roles to get data from different sources. However, since we conducted the study in only one company, we cannot generalize the obtained results without further replication of the study which is discussed as future work in Section 3.8.

With regards to the literature review, the most recent publication was published 2014, which reviewed papers up to 2013. There is a chance that papers that propose newer solutions to our identified challenges have been published since then.

3.6.2 Construct Validity

To minimize this threat we had to make sure that what we wanted to study (Challenges of establishing traceability) was understood by the participants of the study. To achieve this we first had a meeting with the two experts from the two departments where we explained the intentions of the study. In return, they also explained what their departments do. We also sent the interview guide and scope to the participants one week before the study. As mentioned in Section 3.3, the interviews we conducted were not recorded due to legal matters but the interviewer took notes. To make sure that we did not misinterpret our findings, we showed our initial analysis to one of the senior experts for confirmation. This is known as member checking [119].

3.6.3 Reliability

To ensure that the results of a study are reliable it is important to make sure that the study can be repeated by other researchers and get the same results. While the settings of the interview cannot be replicated, the artifacts used such as the definition of the scope of the study and the interview guide were well documented and can be used for replication of the study.

3.7 Related Work

Regan and colleagues [9], conducted a literature review to identify the barriers of traceability and their solutions from literature. In their work, they propose a framework which consists of the categories of the challenges and their solutions. Their framework is quite similar to the categories of challenges that we have proposed. However, their work does not investigate if these proposed solutions work in practice, which is something that our research does by complementing the literature review with an industrial case study.

Further related studies are those by Torkar et al. [10] and Cleland et al [26]. In [10], the authors performed a systematic literature review, with the aim of identifying requirements traceability definitions, tools, practices and challenges. They also complement their work with a case study in two companies. In their results, they give a list of challenges and how they are relevant for the two companies. That study is similar to ours but their literature

review only includes papers up to 2007 while ours includes studies of up to 2014. Also in their research the studied companies are not in the automotive domain but in the telecommunication domain and mobile applications domain. In [26], the authors reviewed four recent industrial studies and interviewed eight practitioners on traceability practices. The authors propose several research questions that need to be investigated in order to achieve the seven desired qualities of traceability proposed in [82]. These qualities are that traceability needs to be purposed, cost-effective, configurable, trusted, scalable, portable and valued. These quality attributes correspond to the findings in our study, for instance for traceability to be trusted, there needs to be methods for assessing the quality of links. Also in the study, one of the conclusions is that more collaboration with industrial practitioners and researchers is needed in order to ensure that the solutions from research are actually applicable in practice. Our study is an example of the research proposed here.

Another study is by Kannenberg & Saiedian [85] where the authors study the existing literature to investigate why software requirements traceability still remains a challenge. They conclude that manual traceability methods and existing tools are inadequate for the needs of the software development companies.

3.8 Conclusion

The aim of this paper was to investigate which traceability challenges exist in the automotive domain and how solutions proposed in literature are applicable for solving these challenges. We conducted a case study with an automotive supplier and a tertiary literature study on traceability research. Our study found that there is a large overlap between the challenges generally reported in literature with those found at the automotive company. 16 of the 19 challenges found in literature were also observed at the company. However, only five of these challenges have been fully solved at the company, five are partially solved while six remain unsolved. The unsolved challenges are; 1) Manual work of creating and maintaining traceability links, 2) Traceability activities perceived as an overhead, 3) Lack of visualization tools, 4) Manual assessment of links, 5) Hard to measure the return on investment of traceability and 6) Lack of universal standards for exchange of traceability links. Based on these unsolved challenges, we can conclude that current traceability practices are costly and inefficient, with a return on investment that remains difficult to prove in practice. For traceability to be widely adopted, there is a need for both researchers and practitioners to investigate effective and cost efficient techniques for traceability.

From the literature, there already exists some solutions proposed for most of the partially solved and unsolved challenges, however, for the case we investigated, these solutions were either tried and did not fully solve the problem (for instance holistic tools to solve the diversity of tools problem) or the solutions could not be applied due to constraints that are specific to the automotive domain such as the requirement to follow safety standards such as ISO 26262 (for instance using machine learning to generate links for safety critical applications). It is therefore important to investigate how the proposed

solutions in literature can be tailored and made applicable to this domain. In cases where tailoring of the solutions will not be enough, new approaches to solve these challenges can be investigated.

In Section 3.5 of this paper, we have made some proposals on how the existing solutions can be extended. For future work, we plan to investigate how such extensions will be able to work in practice, by implementing and trying them with practitioners. As part of our research we have developed an open source traceability tool[5] that allows manual creation of links to arbitrary artifacts. Our concrete plans are to investigate how to combine automatically created links (for instance from model transformations) with manually created links. We will also investigate how to support users with semi-automatic maintenance of traceability links through notifications and collaborative features such as commenting on links. Furthermore, we will investigate how such a dedicated traceability tool can be integrated in the development process of a company. To contribute to the best practices of traceability, we also plan to work together with our industrial partners mainly from the automotive domain to provide different traceability metamodels for the different systems found in this domains. For instance we will provide metamodels for traceability when developing product lines and when developing multi-core systems.

Acknowledgements

This work has been sponsored by an ITEA project, AMALTHEA4Public[6].

[5] https://projects.eclipse.org/projects/modeling.capra
[6] `http://www.amalthea-project.org`

Chapter 4

Paper C

Traceability Maintenance: Factors and Guidelines

S. Maro, A. Anjorin, R. Wohlrab, J.-P. Steghöfer

31st International Conference on Automated Software Engineering (ASE 2016), Singapore, Singapore, September 3-7, 2016.

Abstract

Traceability is an important concern for numerous software engineering activities. Establishing traceability links is a challenging and cost-intensive task, which is uneconomical without suitable strategies for maintaining high link quality. Current approaches to Traceability Management (TM), however, often make important assumptions and choices without ensuring that the consequences and implications for traceability maintenance are feasible and desirable in practice. In this paper, therefore, we identify a set of core factors that influence how the quality of traceability links can be maintained. For each factor, we discuss relevant challenges and provide guidelines on how best to ensure viable traceability maintenance in a practical TM approach. Our results are based on and supported by data collected from interviews conducted with: (i) 9 of our industrial and academic project partners to elicit requirements for a traceability tool, and (ii) 24 software development stakeholders from 15 industrial cases to provide a broader overview of the current state of the practice on traceability maintenance. To evaluate the feasibility of our guidelines, we investigate a set of existing TM solutions used in industry with respect to our guidelines.

4.1 Introduction and Motivation

Traceability can be defined as the ability to relate different artefacts created during the development of a software system. This also includes the ability to identify stakeholders that have contributed to the creation of artefacts, and the rationale that explains the need of these artefacts [6]. Traceability Management (TM) incorporates the creation, maintenance, and use of traceability links. It is an important concern that cuts across numerous domains and application scenarios including tool integration [120], requirements management (RM) [28], software product line management [121, 122], model driven engineering [123–125], and compliance with standards such as CMMI [126] and ISO 26262 [47].

All activities associated with keeping traceability links up to date and consistent are referred to as traceability maintenance. Traceability links rapidly become obsolete and effectively useless if they are not maintained as other artefacts evolve [22]. As the manual maintenance of links is error prone and expensive, a tool-supported approach to traceability maintenance is required if the benefits of traceability are to be realised. The main contribution of this paper are factors that impact traceability maintenance and guidelines on how to address them when designing TM tools. This contribution is based on an analysis of the spectrum of possible solutions extracted from interviews with industry practitioners. A further contribution is an overview of how the guidelines are realised in existing TM tools.

The promised benefits of traceability include improving the quality of software systems by supporting tasks related to maintenance, evolution, documentation, testing, and reuse. Traceability makes these tasks less dependent on individual experts and improves system acceptance by increasing understandability [6, 11, 82] A current challenge is, however, to *cost-effectively* enable these promised benefits [22]. To ensure this, it is of the utmost importance to guarantee that a high traceability link quality can be maintained in the face of changes to connected artefacts.

Traceability link "quality" is typically quantified by a combination of measurable properties including completeness, correctness, accuracy, precision, confidence, etc. [11]. These properties can only be defined precisely in a specific context and will thus be referred to collectively in the following as the general level of *consistency* of all traceability links. There are some automated approaches to maintain consistency, e.g., constraint-based [125], grammar-based [127], or based on machine learning techniques [128]. In practice, however, it often remains unclear why the particular chosen approach is feasible or desirable. The data from our interviews reveals two main challenges: (i) traceability links are still mostly created manually in practice as there is not yet sufficient trust in the quality of automation techniques, and (ii) one must cope with connected but highly *heterogeneous* artefacts across tool boundaries.

When designing and developing a TM solution that combines manual and automated traceability maintenance techniques, multiple factors must be considered. It is crucial to understand their consequences on traceability maintenance. Our aim is to provide a systematic set of guidelines that can be applied when establishing traceability maintenance as a crucial part of a practical TM approach. These guidelines are based on a set of primary factors that influence how the consistency of traceability links can be maintained.

For each factor, we discuss relevant challenges and suggest solution strategies together with their respective consequences and implications. These factors and guidelines constitute a novel contribution based on empirical evidence. Thus, the main research question for this paper is as follows:

What are the primary factors that affect how and to what extent a TM solution can provide traceability maintenance?

We conducted semi-structured interviews with 9 industrial practitioners to elicit requirements for a traceability tool, and with 24 additional software development stakeholders from 15 companies to provide a broader understanding of the practical challenges involved in establishing a viable and flexible TM solution.

The rest of the paper is structured as follows: In Section 4.2 we introduce basic terminology. Our main contribution, important factors and guidelines to consider when addressing traceability maintenance, is presented in Section 4.3. In Section 4.4 we evaluate the feasibility of our guidelines in practice by investigating a set of existing TM solutions with respect to our guidelines. Our paper concludes with an overview of related work in Section 4.5, threats to validity in Section 4.6, and future areas of research in Section 4.7.

4.2 Foundations

As TM is a task that cuts across multiple application domains and technological spaces, our terminology is chosen to be generic enough to incorporate both manual and informal TM strategies. Research on *bidirectional transformations* (bx) has many parallels to TM, including the central concept of consistency for a given set of artefacts, as well as the requirement to be as technology agnostic as possible. Our definitions are thus inspired by work on bx such as [129].

Let us refer to the "things" that we want to work with (modify, trace to and from) as *models*. We do not care what exactly models are (this can be very different from one domain to another), only that they can be modified to result in other models. Let us refer to such a modification as a *delta*. We are also interested in different "kinds" of models, which we shall refer to as *model spaces*. A model space basically groups together all possible states of a kind of model, connected by deltas. Again we do not care how such a model space is exactly induced as there are many ways to do this (using metamodels, constraints, grammars, etc.). Finally, we expect deltas to be composable and that it be possible to get from any model to any other model in the same model space. This is summarised succinctly in the following definition.

Definition 1. *(Model Space) A* model space $\mathcal{M} = (M, \Delta)$ *consists of a set* M *of models, a set* Δ *of deltas, and functions* $src : \Delta \to M$, $trg : \Delta \to M$ *that map a delta to its source and target model, respectively. For* $A, A' \in M$, *we denote* $a \in \Delta$ *as* $a : A \to A'$, *if* $src(a) = A$ *and* $trg(a) = A'$.

Every model space $\mathcal{M} = (M, \Delta)$ *is* connected*:* $\forall A, A' \in M\ \exists a : A \to A'$, *and* reflexive*:* $\exists id : M \to \Delta$, *a function mapping every model* A *to* $id_A : A \to A$, *a special* identity *delta. Finally, deltas can be composed via a binary operator* $; : \Delta \times \Delta \to \Delta$, *which is* associative*:* $(a; a'); a'' = a; (a'; a'')$, $\forall a : A \to A', a' : A' \to A'', a'' : A'' \to A'''$, *and for which identity deltas are* neutral*:* $id_A; a = a = a; id_{A'}$.

Example. As our running example we consider a software development project with (i) requirements in the ReqIF[1] format, (ii) implementation models as UML statecharts, and (iii) tests in form of C code. In terms of model spaces we thus have three model spaces: (i) $\mathcal{M}_{req}$ of all possible requirement models connected by deltas representing all possible changes (e.g., addition, deletion, and all means of editing a requirement), (ii) $\mathcal{M}_{uml}$ of all possible statechart models and deltas on statecharts, and (iii) $\mathcal{M}_c$ of all C programs and deltas on C programs. One could additionally restrict the model spaces to "well-formed" models, e.g., only considering C programs that compile, and requirements/statecharts that comply to the ReqIF/UML metamodel (and all constraints).

The concept of a "traceability link" is relatively difficult to fix, ranging in the literature from typed to untyped, binary to n-ary, and interconnected to isolated. In this paper, we choose not to define what a traceability *link* is but rather to view a distinguished model space as a special kind of *traceability model space* for connecting models from other model spaces. This means that traceability models and deltas are just as simple or as rich as any other models and deltas:

Definition 2. *(Traceability Model) A* traceability *model space is a distinguished model space* $\mathcal{M}_\tau = (M_\tau, \Delta_\tau)$*. Models* $T \in M_\tau$ *are referred to as* traceability models*.*

Example. For our running example, we take a model-based approach, defining a traceability model space $\mathcal{M}_\tau$ via a meta-model with an n-ary traceability "link type" connecting a requirement, multiple states and transitions in a statechart, and multiple tests (files with C code) together. We also decide to connect such traceability links with an association "isRelatedTo", effectively grouping related traceability links. The point here is that traceability models can be chosen to be just as rich as any other model.

The exact manner in which a traceability model "connects" a set of other models is also left open and can range from explicit edges (assuming a graph-based representation of models), implicit connections based on attribute values (IDs), and connections based on auxiliary structures such as tables, etc. For this paper, it suffices to introduce a *consistency function* that hides all such details, and decides how consistent a given set of models together with a connecting traceability model is. We choose to allow *levels* of consistency as opposed to "fully consistent" or "completely inconsistent", as a viable traceability maintenance solution should be able to cope with *partially* consistent models [130]:

Definition 3. *(Consistency Function) Given model spaces* $\mathcal{M}_1 \ldots, \mathcal{M}_n$*, and a traceability model space* $\mathcal{M}_\tau$*, a* consistency function *is a function* $R : M_1 \times \ldots \times M_n \times M_\tau \to [0, 1]$*.*

Example. Given model spaces $\mathcal{M}_{req}, \mathcal{M}_{uml}, \mathcal{M}_c$ and traceability model space $\mathcal{M}_\tau$ from our running example, a consistency function $R : M_{req} \times M_{uml} \times M_c \times M_\tau \to [0, 1]$ can be specified as a pragmatic combination of automated sanity checks and decisions to be made by a domain expert:

[1] Requirements Interchange Format (`omg.org/spec/ReqIF/`)

- *Validity:* $R(m_{req}, m_{uml}, m_c, m_\tau) := 0$ if m_τ is invalid, i.e., does not conform to its metamodel. This means that "broken" traceability links (wrong types, violated multiplicities, etc.) are not to be tolerated.
- *Completeness and Correctness:* if m_τ is valid, then $R(m_{req}, m_{uml}, m_c, m_\tau) := 0.2 \cdot comp + 0.8 \cdot corr$, where *comp* is the number of "covered" requirements, i.e., requirements connected to a traceability link, divided by the total number of requirements, and *corr* is the number of correct traceability links divided by the total number of traceability links. Correctness of a traceability link is manually determined by consulting a domain expert.

Note that this consistency function penalises incorrect links more than missing links, and can be extended analogously to handle uncovered elements also in the statecharts and tests.

Given that it is possible to gauge the consistency of a set of models connected by a traceability model, we can now define the task of *traceability maintenance*. The central idea is to define a traceability maintainer on *deltas* instead of just models, i.e., to supply information on how a set of models has evolved, together with the old traceability model. The task of traceability maintenance is then to compute a suitable delta on the traceability model. This is depicted schematically in Fig. 4.1 as "completing the square" and is subsequently formalised in Def. 4. This differs from general consistency restoration, where the input deltas can also be manipulated [129]. In the case of traceability maintenance, the expectation is that *only* the traceability model is changed.

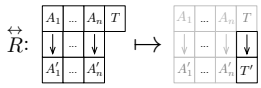

Figure 4.1: Completing the square

Definition 4. *(Traceability Maintainer) Given model spaces* $\mathcal{M}_1 \ldots, \mathcal{M}_n$*, and a traceability model space* $\mathcal{M}_\tau$*, a* traceability maintainer *is a function* $\overleftrightarrow{R}\colon \Delta_1 \times \ldots \times \Delta_n \times M_\tau \to \Delta_\tau$.

Example. For the consistency function R defined previously, the following represents a traceability maintainer $\overleftrightarrow{R}$:

[a] Delete all broken traceability links (fully automatic).
[b] Request a review of all traceability links by a domain expert to evaluate correctness, supplying exactly what was changed as input. Incorrect traceability links that cannot be fixed should be deleted (manual).
[c] Determine uncovered requirements and ask a domain expert to add missing links (semi-automatic).

The assumption in Step (2), which is reasonable but does not hold in general, is that a domain expert (or some software component if this step is automated) does not need to review *all* links if provided with detailed change information.

A basic property of a useful traceability maintainer is that the maintainer either improves (or retains) the current situation, or does nothing at all. This expectation holds for manual and fully/semi-automated traceability maintenance alike; if manipulating the traceability model worsens the current situation then the changes are not worth applying.

Definition 5. *(Consistency Improving) For model spaces $\mathcal{M}_1 \ldots, \mathcal{M}_n$, a traceability model space $\mathcal{M}_\tau$, and a consistency function $R : M_1 \times \ldots \times M_n \times M_\tau \rightarrow [0, 1]$, a traceability maintainer $\overset{\leftrightarrow}{R}$: $\Delta_1 \times \ldots \times \Delta_n \times M_\tau \rightarrow \Delta_\tau$ is* consistency improving *if $R(A_1, \ldots, A_n, T) \leq R(A'_1, \ldots, A'_n, T')$, where $\delta_1 : A_1 \rightarrow A'_1 \in \Delta_1, \ldots, \delta_n : A_n \rightarrow A'_n \in \Delta_n$, and $\delta_T = \overset{\leftrightarrow}{R}(\delta_1, \ldots, \delta_n, T) : T \rightarrow T' \in \Delta_\tau$.*

Further properties concerning, e.g., how "much" of the traceability model is changed (the assumption being that "smaller" changes are preferred), can be specified. The interested reader is referred to, e.g., [131] for a related discussion.

4.3 Influential Factors and corresponding Guidelines

This section presents our findings and discussion based on data collected from the following sources:

(S1) Two focus groups[2] aimed at identifying traceability problems and collecting requirements for a traceability tool from both industrial and academic partners in the Amalthea4public project. The first session was with 5 partners from 2 companies developing embedded systems for forest automotives in Sweden and the second session was with 3 academic partners from 2 universities and one industrial partner (automotive supplier) from Germany. The collected traceability requirements were later refined through phone calls with project partners outside Sweden, and face-to-face meetings with one industrial partner in Sweden.

(S2) Semi-structured interviews[3] with 24 software development stakeholders from 15 industrial cases in Germany and Sweden. This was part of a larger case study aimed at investigating general traceability management practices in industry [132]. For this paper, we only use data related to traceability maintenance collected from the study.

The majority of the cases (cases 1-6) are from the automotive domain, followed by the domains of software development (cases 7-8) and telecommunications (cases 9-10). Other domains are IT services (Case 11), banking self-service automation (Case 12), electrical equipment (Case 13), embedded systems (Case 14), and industrial automation (Case 15). All interviewees had working experience of at least one year in their current roles, including development managers, quality managers, system software architects, and product managers. The interviewees worked in projects varying in size, from four to more than one hundred employees.

The interviews from S1 and S2 were recorded and transcribed. The data was used in several analysis sessions with four researchers to identify key factors and guidelines. We conducted cross case analysis to examine differences between cases and identify practical needs. In the following, we present our findings, referring to respective sources to support our arguments.

[2] `http://tinyurl.com/jotqagy`
[3] `http://tinyurl.com/ht2hmzk`

4.3.1 Factor 1: Versioning

A realistic application scenario will involve *multiple users* working together more or less concurrently on a common set of models. This implies that a *Version Control System* (VCS) of some kind is most probably already present and in use. The conclusion that versioning is a primary factor to consider is supported by (S1), as our project partners require explicitly that versioning be addressed appropriately in a proposed TM solution, and by (S2) as change propagation and VCS solutions play an important role in *all* 15 cases.

The effect of versioning on traceability maintenance can be explained by considering the possible input to a traceability maintainer, depicted in Figure 4.2 with labels ❶, ❷, ❸. If versioning information is completely ignored ❶, traceability maintenance becomes relatively challenging as the maintainer is in effect presented with some versions (perhaps the latest versions) of all models and is expected to update the traceability model. Without any provided deltas, however, a maintainer can only assume that everything was created from scratch meaning that all models, including the traceability model, must be fully inspected. This corresponds to a so called *batch* or non-incremental scenario, well-known from research on model synchronisation (cf. [133] for a classification of application scenarios for model synchronisation). This situation is problematic as it is difficult, if not impossible, to guarantee consistency improvement in any way (the consistency of the previous "state" cannot be determined as previous versions are unavailable). We are ready to formulate our first guideline concerning versioning:

(G1) *Version your traceability model just like all other models, especially ensuring that it is included in any consistent tags (beta, release, etc.). Strive to provide the same level of support and integration with your VCS for your traceability models as for any other models.*

Ample support for (G1) is provided by (S2) as the majority of our interviewees explicitly stated that it would be beneficial to have a versioning solution for traceability models. In some cases (3 of 15), a traceability model is indeed versioned, connecting models in specific versions. This is stated by a software architect from Case 2 to be a major advantage as "correct" traceability links do not get "automatically incorrect", but rather "outdated".

We interpret this as meaning that the task of traceability maintenance is well-defined in the sense that a traceability model T can be evaluated with respect to the correct set of connected models without imposing automatic updates, i.e., forcing a potentially problematic evaluation of T in the context

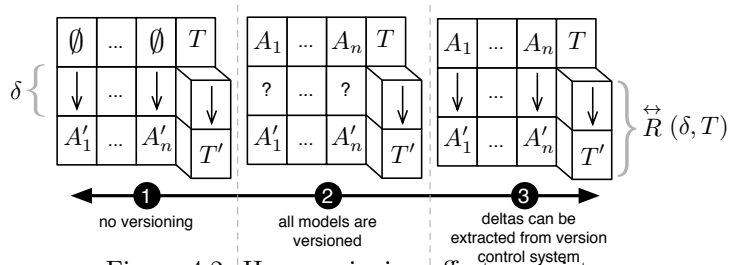

Figure 4.2: How versioning affects consistency

of $A'_1, \cdots, A'_n$ (cf. ❶ in Figure 4.2).

Even if all models are versioned and tagged together with the traceability model, the task of traceability maintenance remains challenging if explicit deltas are not provided. This situation is depicted as ❷ in Figure 4.2 and corresponds to a so-called *state-based* scenario, where the traceability maintainer is only provided with the previous and current versions of all models, and must somehow determine the missing deltas (depicted as question marks in the diagram). This is better than the batch case, but is still suboptimal as the deltas have to be determined, e.g., by comparing the two versions. This is problematic as it entangles consistency improvement with delta recognition, two difficult tasks that should best be handled and controlled separately [129]. This brings us to our second guideline on versioning:

(G2) *Ensure that you are able to extract explicit deltas for all models from your chosen VCS.*

Following this guideline means that you are able to provide all deltas required for traceability maintenance. This is depicted as ❸ in Figure 4.2, representing the ideal situation required for traceability maintenance as all deltas are present.

Guideline (G2) is supported by (S2) as multiple interviewees describe their expectations of how a traceability maintainer should work as follows: a maintainer must check if there are implications caused by *evolving connected models*. If a versioning solution exists, the traceability model must be appropriately *updated with respect to the new versions* of the models, i.e., one must decide if there should (still) be a link or not. Furthermore, the vast majority of interviewees attributed the most common source of inconsistencies to missing "deltas" and corresponding change propagation. For example, a software architect from Case 2 stated that most inconsistencies are probably introduced when performing changes (changing a signal) for which no information is provided about what is connected and potentially affected (e.g., connected requirements). The point here is that without explicit deltas, the entire traceability model must be inspected, regardless of if the consistency maintainer is fully automated, semi-automated, or manual. Even in an optimal situation with all deltas, there is still no general guarantee that necessary updates to the traceability model are "local" in any sense, but the chances of providing an "incremental" and more efficient traceability maintainer are increased.

4.3.2 Factor 2: Tool Boundaries

A typical application scenario for traceability will involve multiple model spaces and often many tools, with which the different models are managed. Planning the scope and boundaries of a TM tool is, therefore, a crucial factor that has a substantial impact on traceability maintenance.

Our requirements (S1) show that project partners require integration of different VCS approaches, RM tools, development environments, and modelling standards, to name just a few of the models spaces and tools involved.

Our interviews (S2) show an equally wide variety of stakeholders from several disciplines, often organised in separate departments with several tools. 14 of our 24 interviewees stated that the heterogeneous nature of the used tools makes traceability management in general, and traceability maintenance in

particular, quite difficult in practice. Due to inadequate tool integration, it is often difficult to establish connections between models stored in different tools.

In some cases, workarounds are provided using the manual mapping of IDs (such as in Case 12 or Case 10). In Case 11, the developers reference the requirements specification with the current version number as a source code comment. Such ad-hoc solutions negatively impact traceability maintenance: the relevant interviewees confirmed that the traceability links established in this manner can only be managed manually and can get easily outdated. It is thus important to plan for a heterogeneous tool landscape, with an understanding of how this impacts traceability maintenance.

The range of choices is depicted schematically in Figure 4.3. On one end of the spectrum ❶ is a *holistic* tool environment that directly supports all relevant tasks, including traceability management. Everything is fully integrated in essentially the same tool. On the other end of the spectrum ❸ is a separate TM tool that only manages traceability models and must establish links to models managed by other tools.

In between these extremes are hybrid solutions ❷ where a mix is chosen. While eliciting requirements from our project partners (S1), we were confronted with contradicting requirements: some partners were interested in traceability to and from primarily requirement specifications and thus suggested that the envisioned TM solution incorporate direct support for RM. This was essentially demanding a hybrid solution combining TM and RM in the same tool. Other partners, however, were already using established RM tools and ruled out changing or switching to a new RM tool.

Interviewees from 6 of our 15 cases from (S2) state that interorganisational collaboration would benefit from using a common tool. For example, a developer from Case 1 stated that it would be helpful to have a common platform to communicate with suppliers. The problem is that there is no standard way to communicate. Different suppliers work with different tools and approaches (e.g., document-based, model-based). With a common tool or platform, one would not have to worry about different standards.

Although a holistic environment might work in some cases — e.g., for smaller companies such as in Case 14, where the collaboration with customers is very close and they can get direct access to development artefacts — in many scenarios this will not be feasible. In Case 6, for example, external collaboration is accomplished using the export of specification documents and e-mail exchange. It was stated that due to legal and intellectual property issues,

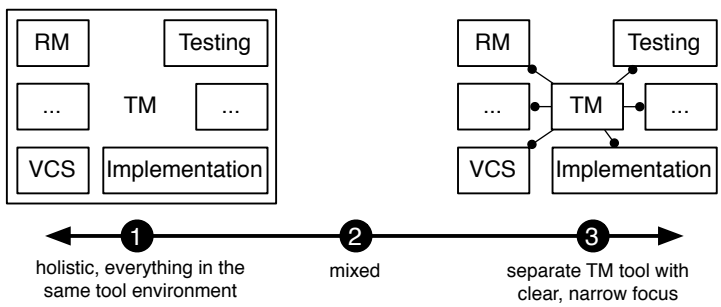

Figure 4.3: Setting tool boundaries

it is infeasible to use one central platform with other organizations. Even in supposedly holistic solutions, such as Case 13 and Case 9, some traceability links still exist that point to bugs or issues reported by the customer and are thus "external" to the tool.

The Head of Software Quality Assurance from Case 12 points out that having an "orthogonal tool that only takes care of traceability" ❸ might be the solution. Such a tool would need to have good interfaces to existing tools (for RM, test management, etc.) and allow the creation of links between models managed by different tools. Our next guideline, therefore, encourages TM solutions to provide direct access to their managed traceability models, as this opens up the TM tool, simplifying integration with external tools:

(G3) *Provide well-defined interfaces and easy, direct access to managed traceability models.*

Holistic and separate TM solutions both have advantages and disadvantages, many of which were stated by our interviewees. With respect to traceability maintenance, a holistic tool environment is able to guarantee a certain minimal consistency, e.g., by forbidding changes that break traceability links, and by forcing the user to first of all delete or adjust affected traceability links before making changes that might cause inconsistencies. One could also weave traceability link creation into a given process supported by the TM tool, ensuring that certain traceability links are created *eagerly*, i.e., "captured" immediately at certain steps in the process. This is advantageous and significantly simplifies the task of traceability maintenance. Establishing such a holistic tool and acquiring adequate acceptance for it is, however, challenging, especially in the context of a multi-partner, open-source project such as Amalthea4public.

For a separate TM tool, every model apart from the traceability model is external in the sense that the TM tool does not control where these models are persisted and how/when they are changed. To enable this, a strategy is required to connect elements in such external models to elements in the traceability model. Possible solutions include establishing *proxies* for these elements, whose creation and maintenance are handled by corresponding (tool) adapters. With respect to consistency, this means that the maintainer has to be able to deal with traceability links that can become broken due to changes (no minimal consistency can be guaranteed), as well as support the *delayed* creation ("recovery") of new traceability links due to changes. This can be substantially more difficult and less scalable than in a holistic situation.

Mixing these strategies, however, tends to amplify the disadvantages, leading to a situation where there is an internal concept not only of traceability models, but also of, for example, requirement models. This means that some import/export mechanism must be provided to get existing requirements specification into the TM tool, paired with the possibility of linking to external models in other tools (e.g., implementation and test models). This results in a complex and potentially confusing workflow, where some models are treated differently than others. In the worst case, imported models might still be changed externally, demanding some additional form of update or synchronisation mechanism. Our guideline in this respect is thus as follows:

(G4) *Aim for either a holistic solution, or a completely separate TM tool with a carefully designed tool adapter concept. Avoid combining both strategies.*

Support for (G4) is provided by our interviews (S2): interviewees from 9 of 15 cases stated that having one common platform across disciplinary borders would be beneficial.

To address the challenges of establishing a separate TM solution and numerous tool adapters, we suggest:

(G5) *Use a common standard and/or technological space as "glue" to simplify the development of tool adapters.*

This is supported by (S2) as, for example, a product manager from Case 5 and a system software architect from Case 6 state that having one tool for all tasks is not feasible but that one should rather try to achieve traceability using better interfaces across tool boundaries (such as OSLC[4] or EMF[5]).

4.3.3 Factor 3: Configurable Semantics

The types and semantics of traceability links vary depending on the domain. It is therefore impossible to implement *generic* but still adequately useful TM consistency functions as consistency is defined based on the type and semantics of the links. This means that "consistency" must be defined and tailored to each domain, possibly even to specific processes, companies, and projects. Defining consistency functions and corresponding traceability maintainers is thus a central and recurring task for TM and should be supported as much as possible by any TM approach.

Support for regarding such diversity as an important factor is provided by (S1): Different project partners require different traceability link types. For instance, partners having a product line approach express the need for links related to variability management while those developing multi-core systems require traceability links for task mapping purposes. Such diverse consistency-related requirements cannot be addressed with a single, fixed definition of a traceability model space, indicating that a suitably flexible configuration process is crucial.

The data collected from our interviews (S2) also reveals a desire for diverse semantics of links e.g., "satisfies", "transferred from", and "refers to" mentioned by a quality manager from Case 7 or "verifies" and "fulfils" mentioned by a system software architect from Case 6, to mention a few.

As motivation for configurable semantics, our interviewees stated: (i) improvement of traceability maintenance (the Head of Software Quality Assurance from Case 12), (ii) to support understandability, especially for new developers (a developer from Case 1), (iii) to simplify reviews and creation of status reports, especially for large models (a project manager from Case 14), and (iv) to enable better search and filter functions (a system software architect from Case 6).

A factor that greatly influences the complexity of consistency functions and corresponding maintainers is the degree to which the traceability model space captures domain-specific semantics and whether the links are explicitly stored or exist implicitly based on, e.g., naming conventions. The spectrum of choices ranges from traceability model spaces without any domain-specific semantics at all (any connection is possible), to traceability model spaces

[4] `http://open-services.net/`
[5] `https://eclipse.org/modeling/emf/`

with a rich domain-specific semantics (connections are restricted to only what makes sense). The former simplifies TM tool support but shifts all complexity to the consistency function and maintainer, while the latter captures some level of consistency already in the traceability models, thereby simplifying corresponding consistency functions and traceability maintainers.

To discuss the effect of implicit vs. explicit semantics on consistency maintenance in more detail, Figure 4.4 depicts the range of choice divided into implicit links ❶, e.g., based on conventions, and explicit links, which are further divided into generic ❷, fixed ❸, and domain-specific ❹.

Implicit links ❶ are connections between traceability models and other models based on conventions such as naming schemes, identifiers, etc. For example, when committing code for a bug fix into a VCS, the ticket number of the bug report should be written in the commit message. Such conventions are problematic as they can be hard to enforce and are often regarded only as "best practice" leading to numerous violations or alternative and possibly conflicting conventions.

Implicit links can be very difficult to check for programmatically, e.g., if the referenced fixed bug is described textually (in a manner that is clear for a human reader) instead of entering its unique ticket number (similar examples could be observed in Case 10, Case 11, and Case 12). Explicit links, represented by elements in the traceability models, are easier to analyse and keep consistent. This brings us to our first guideline on configurable semantics:

(G6) *Avoid implicit, convention-based traceability links and strive instead for explicit links that can be checked with tool support.*

Explicit links can vary substantially regarding the degree to which domain-specific semantics can be captured. *Generic* links ❷, are all of the same basic "type" and can be used to establish connections between anything. This is advantageous for two reasons: (i) it is easy to provide generic tool support, and (ii) such links are flexible in the sense that connections can be established even in unforeseen situations.

From the point of view of consistency maintenance, however, almost all complexity is shifted to the consistency function and maintainer, which have to determine consistency based on the context of established connections. This is not only challenging but can even be impossible in some cases, if there is not enough context information present to retrospectively determine what such a generic link actually means. The disadvantages of generic links can be addressed by allowing additional meta-information to be embedded in links.

In an attempt to retain the advantages of generic links with respect to generic tool support, a common strategy is to provide a rich, but *fixed* traceability model space ❸. This means that the TM solution provides, e.g., numerous

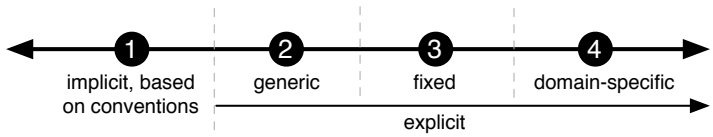

Figure 4.4: How rich are your traceability models?

attributed link types and relations that are, however, fixed with no or only very limited possibilities of extension. This is certainly an improvement over implicit or generic links, but the fixed model space might be either too complex or not rich enough for a certain domain, process, or company. In such a case, complexity is again shifted to the traceability maintainer as it is impossible to embed the required semantics into traceability models. The advantage of this approach is that the TM solution can provide a substantial amount of functionality and consistency checking out-of-the-box.

Finally, explicit links can be completely *domain-specific* ❹ if the TM solution allows the underlying traceability model space to be swapped. This enables domain-specific semantics to be represented, e.g., by appropriate attributes, types and relations. The advantage of this approach is that the TM solution can be adapted to a wide range of domains and applications without sacrificing semantics. Traceability model spaces can be chosen to be very rich, making it impossible, e.g., to create "wrong" links. A challenge with this approach is that generic functionality provided by a TM solution is limited. Substantial effort must be spent on re-implementing domain-specific parts, in particular in relation to traceability maintenance. In practice, therefore, some aspects are typically fixed such as the general "shape" of a traceability link. This discussion is summed up in the following guideline:

(G7) *Prefer domain-specific, semantically rich traceability model spaces as this simplifies traceability maintenance.*

Support from our interviews (S2) is provided by a quality analyst from Case 4, who describes the current usage of generic links as "immature" and "work in progress", and would prefer to be able to attach more semantics. An interesting observation is made by a product manager from Case 5, who mentions that it is virtually impossible to get the exact semantics perfectly right at the start of a project. The semantics must thus be adapted and updated continually during the lifetime of the project, not only by adding new "types" of links, but also by refining and even deactivating existing types. Concerning tool boundaries, the Head of Test Management from Case 7 states that especially links to external tools should be as "rich" as possible. A quality manager from Case 7 describes the current usage of a commercial TM tool with a fixed semantics as unsatisfactory; the TM tool "knows nothing" about extra semantics that users in the company have decided upon and that (hopefully) everyone in the company is aware of and adheres to. Such a fallback to relying on conventions has of course similar disadvantages as using implicit links.

4.3.4 Factor 4: Consistency Specification

Our final factor concerns *how* consistency is specified and consequently maintained. Support for considering this as a primary factor is provided by (S2), as numerous interviewees expressed the need to establish trust in the consistency of traceability links. For example, the Chief Technical Officer from Case 8 stated that the quality of traceability links must be so high that their benefit becomes obvious to all stakeholders. If the stakeholders do not trust the traceability links, then they will not be used, and will not be improved.

The solution space for consistency specification as depicted in Figure 5.1 is spanned by two orthogonal dimensions: a dimension concerning the manner in

which traceability maintainers are applied (the vertical axis), and a dimension characterising the possible classes of their underlying consistency functions (the horizontal axis).

From our interviews, we have identified two main strategies of maintaining consistency: a top-down, process-oriented, mostly eager (consistency violations are fixed immediately) strategy ❶, and a bottom-up, ad-hoc, mostly delayed (consistency violations are fixed only on-demand) strategy ❷. Both strategies appear to be equally successful in practice and are often mixed, with the choice mainly depending on the primary users of the TM solution. We thus suggest the following guideline for this dimension:

(G8) *Ensure that your TM solution supports a flexible combination of both top-down and bottom-up strategies.*

Support for this guideline is mainly provided by (S2): 7 of our 15 cases all have a strong focus on requirements management when it comes to applying traceability. All of these cases except Case 9 use a dedicated RM tool and already organise requirements and their breakdown with it. Integration and system test cases are usually also stored in the tool (or a connected test tool) and linked to the requirements. Many of these cases are part of bigger organizations, in which several companies work together on projects where safety and quality certification is often highly relevant. This demands organised, fixed processes and clear responsibilities. In Case 7, for example, formal reviews are conducted before a milestone is completed. In Case 9, connections between requirements and the design of a measurement system are recorded in spreadsheet files and documents, following a strict and well-defined process. In all these mainly requirements-centred cases, TM in general, and traceability maintenance in particular, is handled in a top-down fashion initiated by project management. Often, but not always, consistency is maintained as part of a defined process, e.g., creating or updating traceability links immediately after the connected artefacts are created or changed in a certain step.

Many other cases feature a more developer-driven, ad-hoc approach. In these cases, traceability links are created and updated on-demand by developers, while focussing on software implementation. This typically involves a software configuration management system handling source code management, tracking of defects (issues, bug reports), and the management of implementation tasks

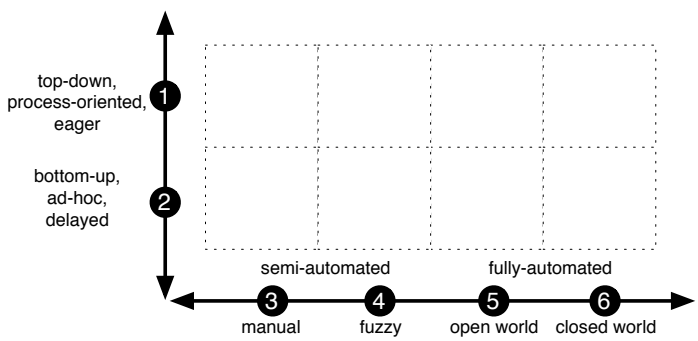

Figure 4.5: How is consistency specified?

(tickets) in a project. These cases typically drive their development based on implementation tasks, and require traceability, e.g., from source code commits to tickets.

There exist some cases that cannot be assigned to exactly one of the two approaches mentioned above. This involves attempts to handle TM in a structured, top-down way — and at the same time, bottom-up approaches invariably arise from the developers' side. In 3 cases, both approaches are clearly present and coexist harmoniously and complementarily. In Case 15, for example, there exists both a requirements-centred approach to connect features and test cases, while a more developer-driven approach is used to link bugs or defects to the respective code commits.

Coming back to the horizontal dimension of Figure 5.1, consistency is in practice often defined manually ❸, i.e., involving a domain expert who has to decide to what extent a given traceability model is consistent or not. As this is, however, quite expensive, considerable research has been conducted with the goal of (fully) automating this task, e.g., [127, 128].

Some of these approaches are fuzzy ❹, i.e., similarity metrics and machine learning techniques are applied to detect patterns and suggest probable connections [128]. Such techniques are often combined with a manual approach resulting in a semi-automatic approach, providing support for a domain expert to make the final decision.

Although semi-automatic approaches already improve the situation, a fully automated approach has the advantage that traceability links can be regarded as "derived", i.e., created on demand and never persisted. This avoids inconsistencies completely by simply re-creating all links as soon as any changes are made. To achieve scalability in cases where this is necessary, numerous incremental and caching strategies can be applied to determine the (potentially) affected set of traceability links and avoid updating or re-creating all links. An underlying assumption that might be necessary, however, is that all traceability links can be (re)created automatically given the current state of all models.

From this discussion, it would appear as though fully-automated strategies are to be clearly preferred; we suggest, nonetheless, taking a *hybrid* solution instead:

(G9) *Support an integrated mix of manual and complementary automated approaches to consistency specification.*

This pragmatic guideline is supported by (S2), indicating that there is simply not yet enough trust and acceptance for full automation. As a quality analyst from Case 4 puts it, automatically creating and updating traceability links is a difficult task; some form of validation or evidence is required to convince users that such traceability links are actually consistent. Especially interviewees from the automotive domain state that it would be very disturbing to have inconsistent traceability links in a system. A manual inspection of traceability links might be expensive, but at least avoids a false sense of high-quality traceability links.

Other interviewees are more open, stating that it would be interesting to improve traceability maintenance via automation strategies; they would, however, still use this more "as an input for a final manual step", as stated by the Chief Technical Officer from Case 8.

There is, at the same time, a clear wish for more automation expressed by numerous interviewees: based on clear naming and structural rules, the Head of Test Management from Case 7 would have no problem maintaining at least a subset of the traceability links automatically. In fact, a team leader from Case 3 mentions basic automation as an important point; simple rules based on steps in a well-defined process *should* ideally be automated to avoid tedious, repetitive, and manual TM-related tasks.

A well-accepted pragmatic strategy of how to combine manual and automated consistency maintenance appears to be the concept of *suspect links* [22, 134]. A product manager from Case 5 states that this technique of applying automatic consistency checks to identify "suspect links" is applied by numerous TM-related tools. Such suspect links are presented to the user for a final decision, possibly together with a set of standard "quick-fix" maintenance strategies that can be applied at the click of a button.

Fully-automated consistency specification approaches can be broadly classified into adhering to either the *Open World Assumption* (OWA) ❺, or the *Closed World Assumption* (CWA) ❻. In an OWA approach, traceability links that the traceability maintainer cannot classify as inconsistent are assumed to be consistent and retained. The maintainer is considered to be incomplete and the language of consistent traceability models is assumed to initially include all possible traceability models (everything would be accepted without a maintainer), and is progressively *restricted* as necessary. Suitable OWA strategies include constraint-based approaches, i.e., providing a set of constraints that must not be violated by any consistent link [125]. Links that are not referenced in any constraint are by default consistent.

In a CWA approach, links that cannot be classified as consistent are assumed to be inconsistent. The maintainer is considered to be complete and the language of consistent traceability models is assumed to be initially empty (everything would be rejected as inconsistent without a maintainer) and is progressively *extended* as necessary. Suitable strategies include generative, grammar-based approaches, i.e., a set of rules that generate all consistent traceability models [127]. Links that cannot be recognised by the maintainer as consistent are by default inconsistent.

Although there are numerous studies on successfully applying machine learning techniques to traceability management and maintenance [128], based on our requirements and interviews, we tend towards CWA approaches:

(G10) *For automatically generated links, prefer no links at all to (possibly) inconsistent links.*

This final guideline is certainly contentious and might not be valid in every application domain, but in the automotive domain and for the development of embedded and safety-critical systems, stakeholders appear to demand both a high confidence in traceability links and a zero tolerance for (possibly) inconsistent traceability links. As a team leader from Case 13 aptly states, users that encounter inconsistent traceability links tend to be utterly confused by connections that do not make sense at all. Having no link at all is actually better than having an inconsistent link — in addition to eventually having to search in some other way for the desired connection, an inconsistent link forces you to first evaluate and conclude that it is indeed inconsistent and unhelpful.

4.4 Strategies in existing TM tools

We now discuss strategies for traceability maintenance employed by three TM tools currently used in industry. The discussion, structured with our proposed guidelines, is based on semi-structured interviews[6] with expert users (in one case) and the developers of the tool (other two cases).

4.4.1 Rational DOORS

IBM Rational DOORS,[7] also known as DOORS Classic, is a requirements management tool that is widely used in industry. It offers traceability features that allow tracing to different types of requirements, e.g., customer requirements, system requirements, software requirements, etc.

Versioning. Both requirements and traceability links are versioned (stored in a database) and can be included in tags. Deltas on requirements are recorded and are available to users. When an artefact connected by a traceability link changes, the user is informed and the delta is presented to the user, who should decide how to update the traceability model. The tool thus adheres to both (G1) and (G2).

Tool Boundaries. As the core functionality of DOORS is RM, traceability links from requirements to requirements are supported out-of-the-box. To address linking to model spaces other than requirements, DOORS provides an OSLC adapter for accessing the requirements. This is currently problematic, as changes to the models in external tools cannot be detected via such OSLC links. The clients of the OSLC adapter provided by DOORS also need to be implemented for each tool which takes substantial effort. Guideline (G3) is followed as links can be accessed via plugins/addons. DOORS, however, does not strictly follow guideline (G4) as it combines RM and TM. Although OSLC has its limitations, (G5) is followed by using OSLC as a common standard for linking to external tools.

Configurable Semantics. The traceability links created with the tool are explicit links, adhering to (G6). Semantics can be configured to a certain extent, as new link types with attributes and restricted source and target types can be created but, for example, n-ary links are impossible. Guideline (G7) is thus followed but with limitations.

Consistency Specification. DOORS allows both top-down and bottom-up consistency management strategies, adhering to (G8). Semi-automatic traceability maintenance is supported by the use of "suspect link detection", i.e., marking a link as "suspicious" as soon as one of the models it connects changes. This is well in accordance with (G9). In addition to manual links, DOORS supports the automatic creation of links for some generated models. For instance, test case skeletons (in form of Excel sheets) can be generated with a link back to the requirements the tests originated from. As the Excel sheets are, however, maintained manually after the generation step they can become inconsistent over time. This means that (G10) is adhered to, but without a viable means of maintaining consistency.

[6] `http://tinyurl.com/htvusac`

[7] `http://www-03.ibm.com/software/products/en/ratidoor`

4.4.2 SystemWeaver

SystemWeaver[8] is a commercial holistic information management solution that aims to support the entire development life-cycle for software and systems engineering. SystemWeaver supports traceability by providing a means of connecting elements of models that reside within the tool.

Versioning. Every model in SystemWeaver is versioned and stored in a common database. The tool has its own VCS and is able to keep track of and provide all deltas. When a model is changed, SystemWeaver checks for any traceability links that are connected to modified elements and imply a potential inconsistency. If such links are found, the relevant deltas are presented to the user who decides if the link is to be updated to point to the newer versions of the models.

With respect to (G1), traceability information is not stored and versioned in an independent traceability model, but as part of the models residing in the tool. The versions of the models containing traceability links implicitly reflect the versions of the traceability links. As the tool implements its own VCS for all models residing in the tool and is able to provide explicit deltas, it adheres quite well to (G2).

Tool Boundaries. SystemWeaver is clearly a holistic tool, adhering to (G4). For models residing in the tool, it is relatively easy to keep track of what has been changed and propose corresponding changes to affected traceability links. It is also possible to configure workflows to prompt the user to create traceability links when certain model elements are created. Since traceability links are parts of the models in the tool, there is a need to traverse the models to get an overview of all traceability links. The tool provides visualization features out of the box and the user is able to get an overview of all the links. Guideline (G3) is thus partly followed. The tool is meant to be sufficient on its own, but some of its customers use it with other tools such as simulation tools. In such cases OSLC is used as glue technology for integration as suggested by (G5). According to an application engineer from the team developing SystemWeaver, ensuring and maintaining consistency to external tools requires a substantial amount of effort. This is done by creating tool-specific adapters that are typically not reusable.

Configurable Semantics. SystemWeaver provides considerable flexibility as metamodels can be used to configure the tool to a particular domain or project. By enabling explicit and domain-specific links, the tool is well in line with both (G6) and (G7). According to SystemWeaver's fixed meta-metamodel, however, the concept of a "connection" is defined as having a single source and a maximum of two targets. An application engineer of the tool stated, however, that this does not appear to be a major limitation for most use cases.

Consistency Specification. SystemWeaver allows the definition of workflows enforcing when links should be updated, as well as ad-hoc link creation. This adheres to (G8) as both top-down and bottom-up approaches are possible.

The tool was originally designed for manual link creation. It is, however, also possible to define rules controlling when and how traceability links are created. Maintenance of links is semi-automatic; when a change is detected in a model element that has a link, the link is flagged as a "suspect link" and

[8] `http://www.systemweaver.se`

the user has to resolve this manually. This is in coherence with (G9). For (G10), as the tool is configurable, it is up to the final user to decide on suitable mechanisms for automatically generating links.

4.4.3 YAKINDU Traceability

YAKINDU Traceability (YT)[9], is a commercial, Eclipse- and EMF-based TM tool. It is a dedicated traceability tool.

Versioning. Traceability models in YT are EMF models, which can be persisted as XML files and thus versioned using any standard VCS. The tool strives to follow (G1) by providing extra diff and merge procedures implemented specially for traceability models.

Concerning (G2), however, version and delta information of the models connected by a traceability link must be obtained from the VCS that is used to store these models. The quality and availability of the deltas thus depend on the chosen VCS. For file types where version and fine-grained delta information cannot be accessed, YT computes a version based on the content of the model. If the model changes, YT analyses the current model and the information stored in the traceability link. If they no longer match, the user is prompted to update the traceability link.

Tool Boundaries. Traceability models are EMF models and can be accessed and used for activities such as impact analysis and change management. This correlates with (G3). YT is a dedicated TM tool, thus adhering to (G4).

All models apart from the traceability model are handled as external models by the tool. To create traceability links to these external models, an EMF representation is required. For non-EMF models, this is handled by tool adapters that create EMF representations of models from different tools. A tool adapter makes a specific type of file format available to YT, for instance, an Excel adapter makes Excel files (up to cell level) available to the tool, while a DOORS adapter makes DOORS requirements available to the tool. This follows (G5), as EMF acts as "glue" technology.

Configurable Semantics. All traceability links are stored in an explicit traceability model (G6). When there is an implicit connection between models (e.g., from one UML component to another), YT provides a rule-based language that can be used to specify how explicit links can be automatically derived. If feasible, such derived links can be stored in memory and not persisted to simplify maintenance.

YT was designed to be a highly configurable tool. Consequently, it must be configured according to the needs of a client. A technical project leader from the company stated that default configurations are not provided as needs differ substantially between companies and even between projects in the same company. Traceability models can be configured for each customer, with a few restrictions concerning the general shape of a link (e.g., a "link" can connect only two things). Another obvious restriction is that all model types to be connected must be supported by corresponding tool adapters. If required, new tool adapters can of course be developed for a customer who is ready to pay for it. YT thus adheres to (G7) with a few constraints.

[9] `http://www.yakindu.de/traceability/`

Consistency Specification. YT does not dictate which approach should be used for traceability maintenance. According to the technical project leader, the tool can be used to support several processes in combination with other tools (G8). In combination with a VCS, for example, YT can enforce access restrictions for editing the traceability model.

Both manual and automatic creation of traceability links is possible (G9). Automatic links are defined by a rule-based language as opposed to employing machine-learning techniques, indicating that (G10) is desired by clients.

4.5 Related Work

Most of the research on traceability maintenance is in the area of (semi-)automatic maintenance of traceability links. These approaches can be categorised as Transformation-Driven, Event-Driven and Rule-Based.

Transformation-Based Approaches take advantage of the fact that (model) transformations can be suitably enriched to additionally produce traceability links. In general, traceability maintenance in this context requires *incremental* transformation approaches, for which the case when both source and target models evolve separately is challenging [33, 34]. This approach also assumes that all artefacts are created via model transformations, which is often not the case in practice (yet). An example for this approach has been proposed by Fockel et al. [125], who describe an approach for semi-automatic establishment and maintenance of traceability links in the automotive domain. Another example is [135], where the authors use a graph-transformation based approach to define, identify, and maintain traceability links.

Event-Driven Approaches leverage events occurring during software development activities to maintain traceability links. As a simple example, deletion of an artefact can be used as a trigger to delete all traceability links connected to it. Research employing this techniques include [136], where a publish and subscribe mechanism is used to connect traceability maintenance tasks to certain events.

Rule-Based Approaches use rules to determine when traceability links should be generated. For example, Spanoudakis et al. [137] define rules based on attributes of artefacts, for creating traceability links between requirements, use cases, and analysis object models. Traceability links are maintained by re-evaluating the rules. Rule-based approaches can be combined with event-driven approaches such as in [22, 112], where traceability maintenance is conducted in two phases: recognising changes based on events, and (re-)evaluating the rules governing link updates.

Even though some of the factors discussed in this paper have been mentioned in the literature (e.g., as requirements in [138]), we are not aware of any categorisation of primary factors together with guidelines for traceability maintenance such as we provide. In an experience report, Kirova et al. [23] propose technology recommendations for a traceability tool. These recommendations support our proposed guidelines especially on versioning and configurable semantics. However, their research was not focused on traceability maintenance and is based on data from only one company. The research by Gotel and Mäder [17] provides guidelines for selecting a TM tool. Their guidelines are

directed at end users, while ours are aimed more at developers of such TM tools.

4.6 Threats to Validity

With regards to data source (S1), the elicited traceability requirements were from industrial and academic partners in the *automotive* industry. It is thus questionable to what extent the requirements can be generalised to other domains.

Data source (S2) was part of a larger traceability study whose main focus was on TM in general. This broader scope could have influenced the interview parts related to traceability maintenance. To minimise this threat, we complemented the data with an analysis of existing TM tools and interviews with TM tool developers and expert users (Section 4.4).

A further threat to the validity of our study is that nine of the interviews from S2 were conducted in German and then translated as accurately as possible to English. For consistency and readability, all interview quotes were also rephrased using our established terminology in the paper (e.g., model instead of artefact, document, or file).

Lastly, researchers' bias in identification of the factors could have affected the results. We mitigated this threat by involving four researchers during analysis and discussing the results with TM tool developers and expert users.

4.7 Conclusion and Future Work

In this paper, we presented factors that greatly influence to what extent a TM solution can support viable traceability maintenance. We suggested guidelines for each factor, which should be followed to avoid potentially negative consequences of certain (combinations of) design decisions.

To evaluate our guidelines, we analysed existing commercial and fairly established or at least reasonably successful TM tools. Our results show that while our guidelines are mostly adhered to (indicating that this is necessary for successful traceability maintenance), configurability and the level of automation can be improved.

Our results and conclusions are backed by interviews with our project partners, a broad range of software development stakeholders, and expert TM tool users and developers.

Our findings can be used by practitioners to develop and select TM tools. Researchers can build up on our findings to create more applicable (automated) TM methods and techniques that take practitioners' needs into consideration.

As future work, we plan to continue ongoing development on an open source TM tool Capra,[10] which will be used and evaluated in the context of the Amalthea4public project. To cater for the wide range of project partners, we aim to address especially configurability (G7) and integrating manual and automation techniques (G9) better than existing TM solutions, applying model-driven technologies to enable truly domain-specific traceability solutions.

[10] `http://salome-maro.github.io/TraceabilityManagement`

Chapter 5

Paper D

Capra: A Configurable and Extendable Traceability Management Tool

S. Maro, J.-P. Steghöfer

24th International Conference on Requirements Engineering (RE2016), Beijing, China, September 12 - 16, 2016.

Abstract

Traceability is a known problem both in academia and industry. One of the main challenges is that there is no one solution that will solve traceability problems for everyone in industry. Traceability needs are dependent on the context of the organization and can differ from project to project in the same organization. To cater for this problem we have developed Capra, an open source, flexible, configurable and extendable traceability management tool. Capra can be tailored according to specific traceability needs of individual projects and organizations.

5.1 Introduction

Traceability in software development refers to the ability to link software artifacts like requirements, code, and tests throughout the development life cycle [6]. Traceability facilitates impact analysis, verifying that requirements have been implemented and tested and in some domains, e.g. in the automotive domain, it is required for fulfillment of safety standards such as ISO 26262 [47]. A major challenge for traceability tool developers is that traceability needs differ from company to company and even from project to project [23, 123]. To build a tool that fits a certain company, one needs to analyze the needs of that company and in most cases the solution will be feasible for that company only. This is not a good business model for commercial tool vendors or open source tool developers who want the same tool to be used in multiple companies. To solve this, a traceability tool needs to be configurable and extendable in such a way that it can be customized specifically to fit the needs of various companies. This is also referred to as *traceability fit for purpose* [82].

We collected requirements for a traceability tool that can integrate into a workflow for the development of embedded systems from a number of industrial partners, mostly in the automotive domain. Based on the collected requirements, Capra[1] has been developed. The choice of which parts of the tool should be configurable is based on the variations that we encountered in the requirements from the different companies. The requirements with the highest priority are the following:

[a] As a user, I want to create traceability links to arbitrary artifacts.
[b] As a project manager, I want to define custom traceability link types for projects.
[c] As a user, I want to visualize artifacts connected by traceability links through a matrix or graph view.

Our current implementation of Capra supports all these requirements by allowing the end user to create, update, and visualize traceability links. It also allows defining custom link types and extending the tool to support arbitrary artifacts.

In comparison to existing tools, Capra supports traceability between arbitrary artifacts (as compared to, e.g., DOORS[2] that, at least off the shelf, only supports traceability between requirements) and a higher degree of customisability (as compared to other tools such as ReqCycle[3] that does not allow modifying storage of traceability links and extending the targets of traceability links easily).

This extended abstract describes the architecture of the tool and an implementation of its default configuration.

5.2 Architectural Design

Capra is an Eclipse plugin and uses the Eclipse Modelling Framework (EMF) as its base technology. It stores the traceability model as an EMF model.

[1] http://salome-maro.github.io/TraceabilityManagement/
[2] `http://www-03.ibm.com/software/products/en/ratidoor`
[3] `http://www.polarsys.org/projects/polarsys.reqcycle`

The tool relies on the Eclipse Extension mechanism[4] and provides extension points for parts of the tool that can be customized. Based on requirements we collected from our project partners, the tool is customisable at three points: i) the types of links to be supported; ii) which types of artifacts can be traced to; and iii) how the links should be stored. Figure 5.1 depicts the extension points and the rationale for each of them is described in the following.

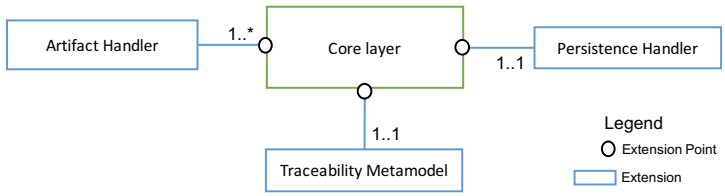

Figure 5.1: Extension points in Capra

5.2.1 Traceability Link Types

Depending on the company, development context, and process used, the traceability links required can differ [82, 139]. For example, traceability links for a company developing web-based solutions are not the same as links for companies developing embedded software. To address the different link types, the tool offers an extension point for the traceability metamodel (see Figure 5.1). Here the end user (company), can define the types of links through a metamodel and supply it to the tool. Examples of link types are "verifies", "implements", "refines", "related to" etc.

5.2.2 Supported Artifact Types

Software development usually involves a number of activities such as requirements engineering, design, implementation and testing. In most cases, each of these activities use different tools and produce artifacts of different formats. A traceability tool needs to ensure that the different formats can be traced to and from. Since different companies use different tools, it is not easy to foresee which formats a traceability tool should support. This problem of diverse artifacts existing in the development environment has been noted by several studies on traceability [6, 121]. Our tool offers an extension point for Artifact Handlers which allows adding artifact formats based on the needs of the end users.

As discussed, Capra stores the traceability links as an EMF model. To be able to support tracing to other formats, EMF representations of these other formats are required. Implementing an extension for a certain format means providing an EMF representation of that format to the tool using the artifact handler extension point.

[4] `https://wiki.eclipse.org/FAQ_What_are_extensions_and_extension_points`

5.2.3 Persistence Extension Point

The storage of traceability links is another factor that can vary depending on company policies or project set-ups. For some cases it makes sense that there is a traceability model per project while in some cases there can be one traceability model for the whole workspace. The extension point Persistence Handler allows defining such storage locations. It will also allow integrating the traceability model with versioning solutions such as EMF Store, CDO or Git.

5.3 Functionalities of Capra — The Default

One of the challenges of creating a configurable tool is that it cannot be used out of the box without a considerable effort going into the configuration first. Since Capra is also very flexible, its core is not usable without any extensions provided to the extension points. To deal with this, we have implemented a default configuration that offers basic extensions to the tool.

Currently, the default configuration of the prototype offers a simple traceability metamodel that supports creation of traceability links that have source and target of any supported artifact type. As shown in Figure 5.2, there are six supported artifact types: Java code (up to method level), C/C++ code (up to function level), files (such as PDF or MS Word), task tickets supported by Mylyn, and test case execution from a continuous integration tool such as Hudson. For storage of the traceability links, the prototype implements an extension to the persistence extension point that stores all links created in the same work space in one folder.

The tool also has functionality for visualization of the traceability links. The links can be visualized in a matrix as well as a graphical format with artifacts represented as nodes and links represented as edges. Figure 5.3 shows such a graphical view extracted from a Heating Ventilation and Air Conditioning (HVAC) system. The example shows a requirement specification about a "blower" connected to a feature represented by a class. The class is connected to a component and a PDF file, and lastly the component is connected to a state machine and another component.

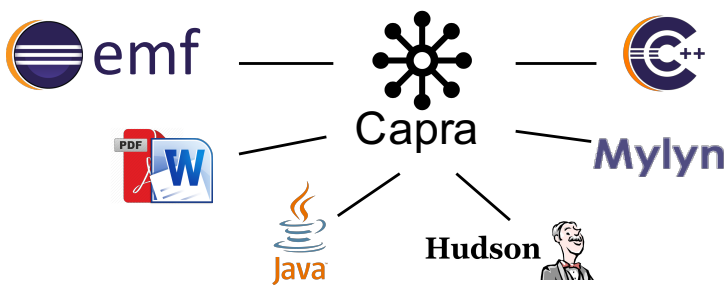

Figure 5.2: Artifact types currently supported by Capra

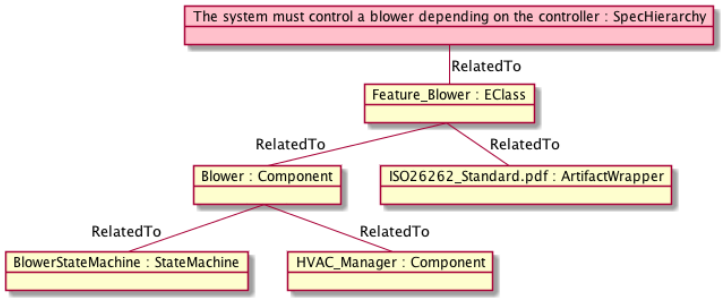

Figure 5.3: Graphical representation of artifacts connected by traceability links

5.4 Conclusions and Future Work

The main contribution of Capra is to provide a configurable and extendable open-source traceability solution. In order to build a flexible tool, one needs to design for flexibility from the start. For future work we aim to incorporate features such as versioning to support trace link maintenance and collaboration features such as discussion, chats and voting in the context of a traceability link in order to improve trace link quality.

Acknowledgment

This work is part of the AMALTHEA4Public project funded by the ITEA Eureka Cluster programme.

Bibliography

[1] D. B. Waghmare, B. V. Arun, K. R. Tiwari, and P. M. Jadhav, "Embedded system & design," 2010.

[2] N. Heumesser and F. Houdek, "Experiences in managing an automotive requirements engineering process," in *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International.* IEEE, 2004, pp. 322–327.

[3] R. Purushothaman and D. E. Perry, "Toward understanding the rhetoric of small source code changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 511–526, 2005.

[4] B. Regnell, R. B. Svensson, and K. Wnuk, "Can we beat the complexity of very large-scale requirements engineering?" in *International Working Conference on Requirements Engineering: Foundation for Software Quality.* Springer, 2008, pp. 123–128.

[5] E. Bouillon, P. Mäder, and I. Philippow, "A survey on usage scenarios for requirements traceability in practice," in *International Working Conference on Requirements Engineering: Foundation for Software Quality.* Springer, 2013, pp. 158–173.

[6] G. Spanoudakis and A. Zisman, "Software traceability: a roadmap," *Handbook of Software Engineering and Knowledge Engineering*, vol. 3, pp. 395–428, 2005.

[7] S. Ibrahim, N. B. Idris, M. Munro, and A. Deraman, "A software traceability validation for change impact analysis of object oriented software." in *Software Engineering Research and Practice*, 2006, pp. 453–459.

[8] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE transactions on software engineering*, vol. 27, no. 1, pp. 58–93, 2001.

[9] G. Regan, F. McCaffery, K. McDaid, and D. Flood, "The barriers to traceability and their potential solutions: Towards a reference framework," in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on.* IEEE, 2012, pp. 319–322.

[10] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. A. Raja, and K. Kamran, "Requirements traceability: a systematic review and industry case

study," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 03, pp. 385–433, 2012.

[11] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. Maletic, and P. Mäder, "Traceability fundamentals," in *Software and Systems Traceability*. Springer, 2012, pp. 3–22.

[12] COEST, "Center of excellence for software traceability (coest)," 2015, accessed : 2016-05-18. [Online]. Available: http://www.coest.org

[13] "Ieee recommended practice for software requirements specifications," *IEEE Std 830-1998*, pp. 1–40, Oct 1998.

[14] J. Radatz, A. Geraci, and F. Katki, "Ieee standard glossary of software engineering terminology," *IEEE Std*, vol. 610121990, no. 121990, p. 3, 1990.

[15] O. C. Gotel and A. C. Finkelstein, "An analysis of the requirements traceability problem," in *Requirements Engineering, 1994., Proceedings of the First International Conference on*. IEEE, 1994, pp. 94–101.

[16] G. Spanoudakis, "Plausible and adaptive requirement traceability structures," in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. ACM, 2002, pp. 135–142.

[17] O. Gotel and P. Mäder, "Acquiring tool support for traceability," in *Software and systems traceability*. Springer, 2012, pp. 43–68.

[18] J. Cleland-Huang, "Just enough requirements traceability," in *Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International*, vol. 1. IEEE, 2006, pp. 41–42.

[19] A. De Lucia, F. Fasano, and R. Oliveto, "Traceability management for impact analysis," in *Frontiers of Software Maintenance, 2008. FoSM 2008.* IEEE, 2008, pp. 21–30.

[20] S. Maro, A. Anjorin, R. Wohlrab, and J.-P. Steghöfer, "Traceability maintenance: factors and guidelines," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2016, pp. 414–425.

[21] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," *Computer*, vol. 40, no. 6, 2007.

[22] P. Mäder, O. Gotel, and I. Philippow, "Enabling automated traceability maintenance through the upkeep of traceability relations," in *Model Driven Architecture-Foundations and Applications*. Springer, 2009, pp. 174–189.

[23] V. Kirova, N. Kirby, D. Kothari, and G. Childress, "Effective requirements traceability: Models, tools, and practices," *Bell Labs Technical Journal*, vol. 12, no. 4, pp. 143–157, 2008.

[24] open services.net, "Open services for lifecycle collaboration," 2017, accessed : 2017-03-20. [Online]. Available: https://open-services.net

[25] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni, "Model traceability," *IBM Systems Journal*, vol. 45, no. 3, pp. 515–526, 2006.

[26] J. Cleland-Huang, O. C. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Software traceability: trends and future directions," in *Proceedings of the on Future of Software Engineering*. ACM, 2014, pp. 55–69.

[27] N. Skrypuch, "Eclipse modeling-emf-home," 2014, accessed : 2014-06-13. [Online]. Available: http://www.eclipse.org/modeling/emf/

[28] O. C. Gotel and A. C. Finkelstein, "An analysis of the requirements traceability problem," in *Requirements Engineering, 1994., Proceedings of the First International Conference on*. IEEE, 1994, pp. 94–101.

[29] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. Maletic, "The grand challenge of traceability (v1. 0)," in *Software and Systems Traceability*. Springer, 2012, pp. 343–409.

[30] A. I. Wasserman, "Tool integration in software engineering environments," in *Software Engineering Environments*. Springer, 1990, pp. 137–149.

[31] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to advanced empirical software engineering*. Springer, 2008, pp. 285–311.

[32] P. S. M. Dos Santos, G. H. Travassos, and M. V. Zelkowitz, "Action research can swing the balance in experimental software engineering," *Advances in Computers*, vol. 83, pp. 205–276, 2011.

[33] I. Galvao and A. Goknil, "Survey of traceability approaches in model-driven engineering," in *Enterprise Distributed Object Computing Conference (EDOC'07)*. IEEE, 2007, pp. 313–324.

[34] I. Santiago, A. Jiménez, J. M. Vara, V. De Castro, V. A. Bollati, and E. Marcos, "Model-driven engineering as a new landscape for traceability management: A systematic literature review," *Information and Software Technology*, vol. 54, no. 12, pp. 1340–1356, 2012.

[35] S. Nair, J. L. de la Vara, and S. Sen, "A review of traceability research at the requirements engineering conference re@ 21," in *Requirements Engineering Conference (RE), 2013 21st IEEE International*. IEEE, 2013, pp. 222–229.

[36] S. Keele, "Guidelines for performing systematic literature reviews in software engineering," in *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. sn, 2007.

[37] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[38] VDA QMC Working Group 13 / Automotive SIG, "Automotive SPICE Process Assessment / Reference Model," Automotive Special Interest Group, Tech. Rep., 2015.

[39] R. Wieringa, "Design science methodology: principles and practice," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 2010, pp. 493–494.

[40] S. Maro and J.-P. Steghöfer, "Capra," 2017, accessed : 2017-03-07. [Online]. Available: https://projects.eclipse.org/projects/modeling.capra

[41] P. Checkland and S. Holwell, "Action research: Its nature and validity," *Systemic Practice and Action Research*, vol. 11, no. 1, pp. 9–21, 1998.

[42] R. K. Yin, *Case Study Research: Design and Methods. 3rd edition.* Thousand Oaks, CA: Sage, 2003.

[43] J. W. Creswell and D. L. Miller, "Determining validity in qualitative inquiry," *Theory into practice*, vol. 39, no. 3, pp. 124–130, 2000.

[44] H. Grönniger, H. Krahn, B. Rumpe, M. Schindler, and S. Völkel, "Textbased modeling," in *4th International Workshop on Software Language Engineering*, 2007.

[45] Plantuml.sourceforge.net, "Plantuml," 2015, accessed : 2015-06-01. [Online]. Available: http://plantuml.sourceforge.net

[46] Eclipse.org, "Graphical editing framework," 2017, accessed: 2017-03-13. [Online]. Available: https://eclipse.org/gef/

[47] International Organization for Standardization, "Road vehicles – functional safety," *ISO26262:2011*, Nov. 2011.

[48] P. Jordan, "Standard iec 62304-medical device software-software lifecycle processes," 2006.

[49] R. F. S. 167, *Software considerations in Airborne Systems and equipment certification.* RTCA, Incorporated, 1992.

[50] R. C. Gronback, *Eclipse modeling project: a domain-specific language (DSL) toolkit.* Pearson Education, 2009.

[51] D. S Wile, "Supporting the DSL spectrum," *CIT. Journal of computing and information technology*, vol. 9, no. 4, pp. 263–287, 2001.

[52] P. Hudak, "Domain-specific languages," *Handbook of Programming Languages*, vol. 3, pp. 39–60, 1997.

[53] www.uml diagrams.org, "Activity diagrams," 2014, accessed: 2015-04-13. [Online]. Available: http://www.uml-diagrams.org/activity-diagrams.html

[54] git scm.com, "Git," 2014, accessed : 2014-06-25. [Online]. Available: http://git-scm.com/

[55] J. H. Bae, K. Lee, and H. S. Chae, "Modularization of the UML metamodel using model slicing," in *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on.* IEEE, 2008, pp. 1253–1254.

[56] S. Efftinge, "Xtext - language development made easy!" 2014, accessed: 2014-06-13. [Online]. Available: http://www.eclipse.org/Xtext/

[57] Eclipse.org, "File:ATL EMFTVM trace.png," 2015, accessed : 2015-05-18. [Online]. Available: https://wiki.eclipse.org/File:ATL_EMFTVM_Trace.png

[58] Gentleware.com, "UML-to-ecore plug-in," 2014, accessed : 2014-06-30. [Online]. Available: http://www.gentleware.com/fileadmin/media/archives/userguides/poseidon_users_guide/ecoreguide.html

[59] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: Eclipse Modeling Framework.* Pearson Education, 2008.

[60] Eclipse.org, "Emf compare - compare and merge your emf models," 2014, accessed : 2014-06-30. [Online]. Available: http://www.eclipse.org/emf/compare/overview.html

[61] A. Bergmayr, M. Wimmer, W. Retschitzegger, and U. Zdun, "Taking the pick out of the bunch-type-safe shrinking of metamodels," *Fachtagung des GI-Fachbereichs Softwaretechnik*, p. 85, 2013.

[62] C. Seybold, M. Glinz, S. Meier, and N. Merlo-Schett, "An effective layout adaptation technique for a graphical modeling tool," in *Proceedings of the 25th International Conference on Software Engineering*, ser. ICSE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 826–827.

[63] P. Stevens, "Bidirectionally tolerating inconsistency: partial transformations," in *Fundamental Approaches to Software Engineering.* Springer, 2014, pp. 32–46.

[64] Github.io, "Textuml toolkit," 2015, accessed : 2015-06-01. [Online]. Available: http://abstratt.github.io/textuml/readme.html

[65] Plaintext.com, "Planttext," 2015, accessed : 2015-06-01. [Online]. Available: http://www.planttext.com

[66] C.-L. Lazăr, "Integrating alf editor with eclipse uml editors." *Studia Universitatis Babes-Bolyai, Informatica*, vol. 56, no. 3, 2011.

[67] M. Scheidgen, "Textual modelling embedded into graphical modelling," in *Model Driven Architecture–Foundations and Applications.* Springer, 2008, pp. 153–168.

[68] F. Jouault and J. Delatour, "Towards fixing sketchy uml models by leveraging textual notations: Application to real-time embedded systems," in *14th International Workshop on OCL and Textual Modeling: Applications and Case Studies.* CEUR-WS, 2014, pp. 73–82.

[69] L. Engelen and M. van den Brand, "Integrating textual and graphical modelling languages," *Electronic Notes in Theoretical Computer Science*, vol. 253, no. 7, pp. 105–120, 2010.

[70] M. Voelter, D. Ratiu, B. Schaetz, and B. Kolb, "mbeddr: an extensible c-based programming language and ide for embedded systems," in *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. ACM, 2012, pp. 121–140.

[71] M. Voelter, J. Siegmund, T. Berger, and B. Kolb, "Towards user-friendly projectional editors," in *Software Language Engineering*. Springer, 2014, pp. 41–61.

[72] JetBrains, "Jetbrains :: Meta programming system - language oriented programming environment and dsl creation tool," 2014, accessed : 2014-06-15. [Online]. Available: http://www.jetbrains.com/mps/

[73] G. Giachetti, B. Marín, and O. Pastor, "Using UML as a domain-specific modeling language: A proposal for automatic generation of UML profiles," in *Advanced Information Systems Engineering*. Springer, 2009, pp. 110–124.

[74] G. Giachetti, M. Albert, B. Marín, and O. Pastor, "Linking UML and MDD through UML profiles: a practical approach based on the UML association." *J. UCS*, vol. 16, no. 17, pp. 2353–2373, 2010.

[75] S. Walderhaug, E. Stav, and M. Mikalsen, "Experiences from model-driven development of homecare services: UML profiles and domain models," in *Models in Software Engineering*. Springer, 2009, pp. 199–212.

[76] A. Abouzahra, J. Bézivin, M. D. Del Fabro, and F. Jouault, "A practical approach to bridging domain specific languages with UML profiles," in *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA*, vol. 5. Citeseer, 2005.

[77] M. Wimmer, "A semi-automatic approach for bridging DSMLs with UML," *International Journal of Web Information Systems*, vol. 5, no. 3, pp. 372–404, 2009.

[78] G. Giachetti, B. Marín, and O. Pastor, "Using UML profiles to interchange DSML and UML models," in *Research Challenges in Information Science, 2009. RCIS 2009. Third International Conference on*. IEEE, 2009, pp. 385–394.

[79] M. Broy, "Challenges in automotive software engineering," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 33–42.

[80] J. Dannenberg and J. Burgard, "Car innovation: A comprehensive study on innovation in the automotive industry," 2015.

[81] A. Busnelli, "Car Software: 100M Lines of Code and Counting," https://www.linkedin.com/pulse/20140626152045-3625632-car-software-100m-lines-of-code-and-counting, 2014, [Online; accessed 07-10-2016].

[82] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, and G. Antoniol, "The quest for ubiquity: A roadmap for software and systems traceability research," in *Requirements Engineering Conference (RE), 2012 20th IEEE International*. IEEE, 2012, pp. 71–80.

[83] C. Lee, L. Guadagno, and X. Jia, "An agile approach to capturing requirements and traceability," in *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE 2003)*, 2003.

[84] F. Blaauboer, K. Sikkel, and M. N. Aydin, *Deciding to Adopt Requirements Traceability in Practice*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 294–308.

[85] A. Kannenberg and H. Saiedian, "Why software requirements traceability remains a challenge," *CrossTalk The Journal of Defense Software Engineering*, vol. 22, no. 5, pp. 14–19, 2009.

[86] S. Maro, M. Staron, and J.-P. Steghöfer, "Challenges of establishing traceability in the automotive domain," in *Proceedings of the 9th International Conference on Software Quality*. Springer, 2017.

[87] I. H. Krüger, E. C. Nelson, and K. V. Prasad, "Service-based software development for automotive applications," SAE Technical Paper, Tech. Rep., 2004.

[88] A. Pretschner, M. Broy, I. H. Kruger, and T. Stauner, "Software engineering for automotive systems: A roadmap," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 55–71.

[89] K. Grimm, "Software technology in an automotive company: major challenges," in *Proceedings of the 25th international conference on Software Engineering*. IEEE Computer Society, 2003, pp. 498–503.

[90] C. Salzmann and T. Stauner, "Automotive software engineering," in *Languages for system specification*. Springer, 2004, pp. 333–347.

[91] J. Mossinger, "Software in automotive systems," *IEEE software*, vol. 27, no. 2, p. 92, 2010.

[92] D. Tang and X. Qian, "Product lifecycle management for automotive development focusing on supplier integration," *Computers in industry*, vol. 59, no. 2, pp. 288–295, 2008.

[93] G. Volpato, "The oem-fts relationship in automotive industry," *International Journal of Automotive Technology and Management*, vol. 4, no. 2-3, pp. 166–197, 2004.

[94] P. Rempel and P. Mäder, "A quality model for the systematic assessment of requirements traceability," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. IEEE, 2015, pp. 176–185.

[95] S. F. Königs, G. Beier, A. Figge, and R. Stark, "Traceability in systems engineering–review of industrial practices, state-of-the-art technologies and new research solutions," *Advanced Engineering Informatics*, vol. 26, no. 4, pp. 924–940, 2012.

[96] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th international conference on evaluation and assessment in software engineering*, vol. 17, no. 1. sn, 2008, pp. 1–10.

[97] M. F. Bashir and M. A. Qadir, "Traceability techniques: A critical study," in *2006 IEEE International Multitopic Conference*. IEEE, 2006, pp. 265–268.

[98] B. Ramesh, "Factors influencing requirements traceability practice," *Communications of the ACM*, vol. 41, no. 12, pp. 37–44, 1998.

[99] S. Winkler and J. Pilgrim, "A survey of traceability in requirements engineering and model-driven development," *Software and Systems Modeling (SoSyM)*, vol. 9, no. 4, pp. 529–565, 2010.

[100] M. A. Javed and U. Zdun, "A systematic literature review of traceability approaches between software architecture and source code," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2014, p. 16.

[101] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang, "Strategic traceability for safety-critical projects," *IEEE software*, vol. 30, no. 3, pp. 58–66, 2013.

[102] A. Von Knethen and B. Paech, "A survey on tracing approaches in practice and research," *Frauenhofer Institut Experimentelles Software Engineering, IESE-Report No*, vol. 95, 2002.

[103] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010, pp. 155–164.

[104] M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1565–1616, 2014.

[105] I. Galvão and A. Goknil, "Survey of traceability approaches in model-driven engineering," *Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOC*, pp. 313–324, 2007.

[106] R. M. Parizi, S. P. Lee, and M. Dabbagh, "Achievements and challenges in state-of-the-art software traceability between test and code artifacts," *IEEE Transactions on Reliability*, vol. 63, no. 4, pp. 913–926, 2014.

[107] P. Rempel, P. Mäder, T. Kuschke, and I. Philippow, "Requirements traceability across organizational boundaries-a survey and taxonomy." in *REFSQ*. Springer, 2013, pp. 125–140.

[108] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010, pp. 95–104.

[109] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 16, no. 4, p. 13, 2007.

[110] G. Spanoudakis, A. Zisman, E. Pérez-Minana, and P. Krause, "Rule-based generation of requirements traceability relations," *Journal of Systems and Software*, vol. 72, no. 2, pp. 105–127, 2004.

[111] D. Cuddeback, A. Dekhtyar, and J. H. Hayes, "Automated requirements traceability: The study of human analysts," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*. IEEE, 2010, pp. 231–240.

[112] P. Mäder and O. Gotel, "Towards automated traceability maintenance," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2205–2227, 2012.

[113] A. Egyed, P. Grünbacher, M. Heindl, and S. Biffl, "Value-based requirements traceability: Lessons learned," in *Design requirements engineering: a ten-year perspective*. Springer, 2009, pp. 240–257.

[114] V. Gaur and A. Soni, "A fuzzy traceability vector model for requirements validation," *International Journal of Computer Applications in Technology*, vol. 47, no. 2-3, pp. 172–188, 2013.

[115] J. Cleland-Huang, C. K. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 796–810, 2003.

[116] J. Cleland-Huang, G. Zemont, and W. Lukasik, "A heterogeneous solution for improving the return on investment of requirements traceability," in *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*. IEEE, 2004, pp. 230–239.

[117] P. Arkley and S. Riddle, "Overcoming the traceability benefit problem," in *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*. IEEE, 2005, pp. 385–389.

[118] O. Pedreira, F. García, N. Brisaboa, and M. Piattini, "Gamification in software engineering–a systematic mapping," *Information and Software Technology*, vol. 57, pp. 157–168, 2015.

[119] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *Software Engineering, IEEE Transactions on*, vol. 25, no. 4, pp. 557–572, 1999.

[120] B. Aichernig, K. Hormaier, F. Lorber, D. Nickovic, R. Schlick, D. Simoneau, and S. Tiran, "Integration of requirements engineering and test-case generation via OSLC," *14th International Conference on Quality Software*, pp. 117–126, 2014.

[121] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J. C. Royer, A. Rummler, and A. Sousa, "A model-driven traceability framework for software product lines," *Software and Systems Modeling*, vol. 9, no. 4, pp. 427–451, 2010.

[122] L. Lamb, W. Jirapanthong, and A. Zisman, "Formalizing traceability relations for product lines," in *6th Int. Workshop on Traceability in Emerging Forms of Software Engineering.* ACM, 2011, p. 42.

[123] S. Winkler and J. von Pilgrim, "A survey of traceability in requirements engineering and model-driven development," pp. 529–565.

[124] S. Walderhaug, U. Johansen, E. Stav, and J. Aagedal, "Towards a generic solution for traceability in MDD," in *ECMDA Traceability Workshop*, 2006, pp. 41–51.

[125] M. Fockel, J. Holtmann, and J. Meyer, "Semi-automatic establishment and maintenance of valid traceability in automotive development processes," *2nd Int. Workshop on Software Engineering for Embedded Systems (SEES'12)*, pp. 37–43, 2012.

[126] CMMI Product Team, "CMMI for Development, version 1.3," Software Engineering Institute, Tech. Rep. CMU/SEI-2010-TR-033, November 2010.

[127] A. Seibel, R. Hebig, and H. Giese, "Traceability in model-driven engineering: Efficient and scalable traceability maintenance," 2012, pp. 215–240.

[128] A. de Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk, "Information retrieval methods for automated traceability recovery," 2012, pp. 71–98.

[129] Z. Diskin, Y. Xiong, K. Czarnecki, H. Ehrig, F. Hermann, and F. Orejas, "From state- to delta-based bidirectional model transformations: the symmetric case," in *MODELS'11*, ser. LNCS, J. Whittle, T. Clark, and T. Kühne, Eds., vol. 6981. Springer, 2011, pp. 304–318.

[130] P. Stevens, "Bidirectionally tolerating inconsistency: Partial transformations," in *International Conference on Fundamental Approaches to Software Engineering (FASE'14)*, ser. LNCS, S. Gnesi and A. Rensink, Eds., vol. 8411. Springer, 2014, pp. 32–46.

[131] J. Cheney, J. Gibbons, J. McKinna, and P. Stevens, "Towards a principle of least surprise for bidirectional transformations," in *BX 2015*, ser. CEUR Workshop Proceedings, A. Cunha and E. Kindler, Eds., vol. 1396. CEUR-WS.org, 2015, pp. 66–80.

[132] R. Wohlrab, J.-P. Steghöfer, E. Knauss, S. Maro, and A. Anjorin, "Collaborative traceability management: Challenges and opportunities," in *Proceedings of 24th IEEE International Requirements Engineering Conference (RE' 16)*, 2016, p. 10.

[133] Z. Diskin, A. Wider, H. Gholizadeh, and K. Czarnecki, "Towards a rational taxonomy for increasingly symmetric model synchronization," in *International Conference on Theory and Practice of Model Transformations (ICMT 2014)*, ser. LNCS, D. D. Ruscio and D. Varró, Eds., vol. 8568. Springer, 2014, pp. 57–73.

[134] N. Drivalos-Matragkas, D. S. Kolovos, R. F. Paige, and K. J. Fernandes, "A state-based approach to traceability maintenance," in *Proceedings of the 6th ECMFA Traceability Workshop*. ACM, 2010, pp. 23–30.

[135] H. Schwarz, J. Ebert, and A. Winter, "Graph-based traceability: a comprehensive approach," *Software & Systems Modeling*, vol. 9, no. 4, pp. 473–492, 2010.

[136] J. Cleland-Huang, C. K. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," *Transactions on Software Engineering*, vol. 29, no. 9, pp. 796–810, 2003.

[137] G. Spanoudakis, A. Zisman, E. Pérez-Minana, and P. Krause, "Rule-based generation of requirements traceability relations," *Journal of Systems and Software*, vol. 72, no. 2, pp. 105–127, 2004.

[138] I. Pete and D. Balasubramaniam, "Handling the differential evolution of software artefacts: A framework for consistency management," in *22nd Int. Conf. on Software Analysis, Evolution, and Reengineering (SANER'15)*, 2015, pp. 599–600.

[139] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *TSE*, vol. 27, no. 1, pp. 58–93, 2001.